Ion Diamandi Gheorghe Vass

LOGO

o nouă metodă de a învăța cu ajutorul calculatorului

EDITURA PACIFIC

BUCUREȘTI - 1991

ION DIAMANDI GHEORGHE VASS

LOGO
o nouă metodă de a învăța cu ajutorul calculatorului

EDITURA PACIFIC

BUCUREȘTI
1991

Desene: EMIL BOJIN

I.S.B.N. 973—95036—1—6

CUVÎNT ÎNAINTE

Limbajul LOGO a fost creat de Seymour Papert și colaboratorii săi de la Massachusetts Institute of Technology (MIT), care au pornit de la teoria învățării și de la ideile lui Piaget privind reorientarea educației pe baza psihologiei copilului. Sigur, se pune întrebarea privitoare la motivul care a stat la baza creării unui nou limbaj de programare în momentul în care exista, deja, la dispoziția utilizatorilor o multitudine de limbaje de programare. Răspunsul este legat de faptul că fiecare limbaj de programare, prin facilitățile și lipsurile sale, favorizează formarea și utilizarea unui stil propriu de a realiza programe, dar nici unul nu reprezintă o bază optimă care să stimuleze la maxim calitățile care sălășluiesc în stare latentă la copii, cum sînt : capacitatea de modelare și rezolvare de probleme și spiritul de explorare. În plus, unele din limbajele de programare alese pentru inițiere în informatică (datorită comodității lor în utilizare) conduc către un stil în disonanță cu stilurile moderne de programare, clare și eficiente. Limbajul LOGO, în schimb, are încorporate toate conceptele moderne care s-au impus în ultimii ani în informatică. Astfel, LOGO permite, pe lîngă uzualele calcule aritmetice, și manipularea cu ușurință și naturalețe a cuvintelor și frazelor, fiind astfel adaptat explorării limbajelor naturale și artificiale. În plus, o mare parte a succesului limbajului LOGO rezidă în facilitățile sale grafice. Aceste facilități cunoscute sub numele de „TURTLE GRAPHICS" sînt implementate în toate limbajele moderne și permit atît realizarea de către începători (chiar copii mici) a unor desene, cît și explorarea unor concepte de matematică avansată (analiză, topologie, algebră, geometrie diferențială, mecanică clasică și chiar relativistă etc.).

Prin succesul înregistrat, prin răspîndirea sa din ce în ce mai mare, prin revoluționarea sistemelor instrucționale, LOGO a depășit granițele unui limbaj obișnuit de programare, devenind un nou stil, o nouă metodă de gîndire și de învățare.

Lucrarea de față își propune în primul rînd tocmai familiarizarea cu acest nou mod de gîndire și învățare. În acest scop, în primele capitole se realizează prezentarea și introducerea unor noțiuni fundamentale legate de calculatoare și informatică. Introducerea în domeniu se adresează atît celor neinițiați, pentru care LOGO reprezintă un prim contact cu calculatorul, cît și celor care au luat cunoștință cu calculatorul prin intermediul limbajului BASIC. Astfel, lucrarea de față se înscrie pe linia ciclului deschis, în anul 1988, prin lucrarea „Partenerul meu de joc—calculatorul“ editată de RECOOP.

După explicarea unor noțiuni ca: ecran, tastatură, cursor, pixel, comandă etc. se trece la prezentarea tehnicilor de programare, în care ideea de procedură joacă un rol central. Se prezintă structurile repetitive și cele condiționate, precum și folosirea variabilelor. Se ajunge apoi la tehnici avansate de programare care includ recursivitatea și lucrul cu liste. Nu mai puțin importante în cadrul lucrării sînt capitolele care prezintă aplicații ale tehnicilor învățate: realizarea de modele grafice, rezolvări de diverse probleme, abordarea proiectelor, jocuri. În final se prezintă un memorator care conține lista tuturor comenzilor și operațiilor LOGO pe care le pune la dispoziție versiunea cea mai utilizată în țară, și anume, cea disponibilă pe casetă LOGO (RECOOP-ITCI) și funcțională pe calculatoare compatibile Sinclair Spectrum.

Esențial pentru lucrare este modul specific stilului LOGO, în care se desfășoară introducerea și învățarea limbajului propriu-zis. Lucrarea este concepută astfel încît procesul să se desfășoare prin crearea unui cadru de învățare în care prin problemele și exercițiile propuse spre rezolvare, subiectul să descopere singur legi și tehnici. Chiar din acest motiv nu sînt indicate, de obicei, răspunsurile și rezolvările (și există o multitudine de rezolvări posibile prin care se poate ajunge la un anumit adevăr) lăsîndu-se, astfel, toată libertatea în alegerea formelor concrete pe care le vor îmbrăca acestea.

Această concepție își găsește, poate, cea mai bună exemplificare în capitolul „Probleme școlare“ în care se pun la dispoziția profesorilor din școlile gimnaziale cîteva exemple pentru o metodă eficientă de predare a științelor exacte. Metoda se bazează pe faptul că principala cauză a slabei consolidări actuale a conceptelor de bază din științele exacte este insuficienta experiență proto-științifică pe baza căreia se încearcă introducerea acestor concepte și care poate fi substanțial îmbunătățită prin utilizarea lui LOGO. Se observă că în cadrul acestui capitol, inițierea nu are loc pornindu-se de la „principii“ sau „axiome“, urmîndu-se ordinea deducției logice pînă la teoremele sau aplicațiile de mare importanță. (A se vedea în acest sens modul de abordare pentru demonstrarea teoremei lui Pitagora).

Sperăm astfel că lucrarea va putea reprezenta, atît pentru profesori, cît și pentru copii, o deschidere pentru un nou mod de a învăța, un nou mod de a gîndi.

Autorii

INSTALAREA, TASTATURA ȘI ECRANUL

Configurație necesară

Pentru utilizarea lui LOGO este necesară următoarea configurație:
— un calculator personal compatibil cu tipul Sinclair Spectrum (CIP, JET, HC, TIM-S, COBRA) cu o memorie internă de 48 Ko;
— un televizor sau monitor alb negru sau color (acesta din urmă fiind de preferat, LOGO avînd comenzi speciale pentru lucrul cu culori);
— un casetofon uzual, de exemplu Elektronika-302;
— caseta magnetică LOGO, realizată de **RECOOP** în colaborare cu **ITCI.**

Instalarea programului LOGO

După ce s-a asigurat cuplarea televizorului și a casetofonului, se va putea încărca programul LOGO în memoria calculatorului cu comanda **LOAD" " sau LOAD „LOGO"** (vezi fig. 1).

Dacă după parcurgerea înregistrării va apărea mesajul „TAPE LOADING ERROR", se va putea repeta operația sau se va putea încărca **LOGO** de pe fața a 2-a a aceleiași casete.

Veți putea începe să lucrați cu **LOGO** atunci cînd pe ecran va apărea mesajul „BINE AȚI VENIT ÎN LOGO" și semnul întrebării care simbolizează faptul că se așteaptă introducerea unei comenzi **LOGO.**

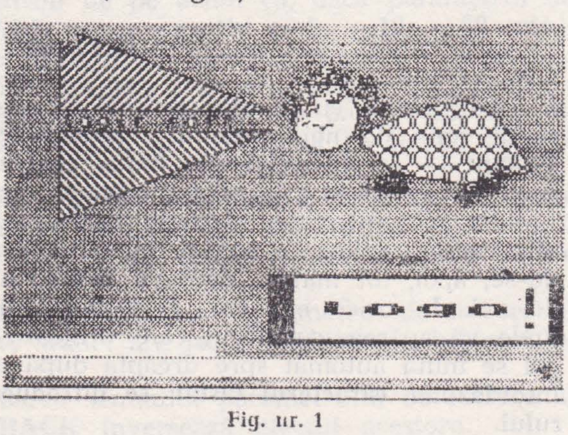


Fig. nr. 1

Dacă doriţi să lucraţi şi cu comenzi în limba română (indicat pentru copiii mici) se poate încărca următorul fişier de pe faţa 1 a casetei (ROMANA) care conţine un set de comenzi în limba română. Încărcarea acestor comenzi se face cu comanda **LOAD "ROMANA**. Pentru informaţii suplimentare puteţi consulta broşura care însoţeşte caseta LOGO realizată de RECOOP.

După încărcarea comenzilor în limba română se va putea lucra cu LOGO atît cu comenzile iniţiale (în limba engleză), cît şi cu comenzile din limba română.

Pentru a se putea lucra cu LOGO, sînt necesare cîteva indicaţii cu privire la tastatura şi ecranul calculatorului.

Tastatura

Calculatorul este prevăzut cu o mulţime de butoane numite **taste**; locul unde se află acestea se numeşte **tastatură**.

Pe fiecare tastă se află o literă sau o cifră ; apăsarea tastei face să apară pe ecran litera sau cifra respectivă. „Spaţiul liber" sau mai pe scurt „Spaţiul" (dintre două cuvinte, de exemplu) este considerat şi el ca un semn (caracter) ; tasta corespunzătoare are pe ea inscripţia SPACE — care înseamnă în engleză tocmai „spaţiu".

Caracterele speciale (+ — / * ? = ; şi altele) apar, fiecare, în colţul din dreapta sus al unei taste obişnuite. Pentru a face ca un astfel de semn să apară pe ecran, este necesar să apăsăm tasta respectivă, dar **ţinînd apăsată în acelaşi timp şi tasta SS**. Tasta **SS** se numeşte **SYMBOL SHIFT**, ceea ce înseamnă „trecerea la simbol".

Majusculele (literele mari) apar pe ecran dacă, în timp ce se tastează litera respectivă, se **ţine apăsată şi tasta CS**. Tasta **CS** se numeşte **CAPS SHIFT**; „caps" este prescurtarea de la „capitals", care înseamnă „majuscule".

Puteţi să exersaţi tastînd cu un spaţiu între ele grupuri de cîteva litere, cifre, caractere speciale şi majuscule.

Ecranul

Să privim cu atenţie ecranul ; pe el se vede, la începutul rîndului scris de noi, „semnul întrebării" („?"). El se mai numeşte **prompt** — acest cuvînt din limba engleză înseamnă „gata", adică, în cazul nostru, „sînt gata pentru comanda următoare".

La sfîrşitul rîndului scris apare un pătrat negru clipitor, numit **cursor**; ştim că, iniţial, cursorul s-a aflat lîngă prompt, mutîndu-se, apoi, tot mai la dreapta, pentru a face loc caracterelor tastate de noi. În orice moment, deci, cursorul arată poziţia de pe ecran unde va apărea caracterul care urmează a fi tastat ; deoarece cursorul se mută automat spre dreapta după orice tastare, putem spune că întotdeauna **caracterul tastat se introduce înaintea (la stînga) cursorului.**

Vom cunoaște, acum, manevrele de corectare a textelor de pe ecran.

Ștergerea unui caracter aflat la stînga cursorului (adică înainte de acesta) se realizează **ținînd apăsată tasta CS** și apăsînd (scurt) **tasta 0 (zero)**; de altfel, pe aceasta din urmă găsim scris și cuvîntul **DELETTE** care înseamnă „**șterge**". Acum puteți, de exemplu, să ștergeți de pe ecran ultimul grup de caractere tastat anterior.

Evident, pentru a efectua o modificare (ștergerea sau introducerea de caractere) **în interiorul** unui text oarecare, va trebui să „ducem" mai întîi cursorul la dreapta poziției respective și abia apoi să realizăm ștergerea sau introducerea caracterelor dorite.

Deplasarea cursorului în interiorul unui text, pe un rînd al ecranului, se face cu manevrele :

CS + **5** (acționarea în același timp a tastelor CS și 5) pentru cursor la stînga (←);

CS + **8** pentru cursor la dreapta (→).

Aceste manevre deplasează cursorul prin text, dar niciodată în afara acestuia !

Puteți să vă antrenați ștergînd de pe ecran primul grup de caractere și apoi să corectați celelalte grupe astfel încît să conțină cîte cinci caractere de același tip.

„Textul" astfel obținut depășește un rînd al ecranului, acest lucru este semnalat de calculator prin așezarea semnului de „continuare" („!") la sfîrșitul rîndului care se continuă.

Într-un text care se continuă pe mai multe rînduri, cursorul poate fi deplasat și „pe verticală", de la un rînd la altul prin manevrele :

CS + **6 cursor în jos (↓)**;

CS + **7 cursor în sus (↑)**.

Probabil că ați remarcat, în colțul din dreapta jos al ecranului, o literă stingheră (**„l"**) ; această literă **reprezintă indicatorul de regim al tastaturii.** Tastatura poate fi într-unul din următoarele regimuri :

l — pentru scrierea normală, cu **litere** mici (l reprezintă prima literă a cuvîntului „letters" care în limba engleză înseamnă „litere") ;

C — pentru scrierea cu majuscule (C reprezintă prima literă a cuvîntului „capitals" — litere mari) ;

E — pentru scrierea „extinsă", folosită în **LOGO** doar pentru unele efecte speciale.

Trecerea tastaturii dintr-un regim în altul se poate face cu următoarele manevre :

regim **„l"** ←→ regim **„C"** : **CS** + 2

regim **„l"** sau **„C"** ←→ **regim „E" : CS** + **SS**

Deci, pentru trecerea din regim *„l"* în regim **„C"** și invers, se vor acționa împreună tastele **CS** și 2.

Toate comenzile LOGO se scriu cu litere mari, tastînd literă cu literă ; din acest motiv, **la începerea lucrului cu LOGO,** se **va trece mai întîi tastatura în regim „C".**

BROASCA ŢESTOASĂ

Convenţia broaştei ţestoase

Învăţarea limbajului **LOGO** este un proces activ, bazat pe principiul **„înveţi învăţînd pe altul"**; noi ne vom ocupa de instruirea unei mici **broaşte ţestoase**, aflate pe ecran. Am făcut cunoştinţă cu acest simpatic personaj chiar de la început, imaginea apărînd pe ecran în timpul încărcării programului **LOGO.** Broasca ţestoasă se poate deplasa conform indicaţiilor noastre, iar „urma" lăsată pe ecran va constitui realizarea noastră grafică.

Broscuţa ţestoasă nu face nimic fără să fie îndrumată de noi, dar execută **întocmai** ceea ce îi comandăm; este necesar, deci, să ştim ce comenzi „înţelege" ea şi să folosim aceste comenzi, astfel încît să execute cu fidelitate desenul dorit (proiectat) de noi.

De ce s-a ales tocmai o broscuţă ţestoasă pentru a fi instruită? Pentru simplul fapt că aceasta nu este niciodată grăbită, dar este meticuloasă, perseverentă, executînd pe rînd toate operaţiile necesare atingerii scopului propus.

În limba engleză broasca ţestoasă este numită printr-un singur cuvînt: TURTLE; de aceea, şi noi vom folosi de multe ori un singur cuvînt (fie „broasca", fie „ţestoasa") pentru a ne referi la colaboratoarea noastră.

Comenzile LOGO se dau prin tastarea unor cuvinte din limba engleză sau prin prescurtările acestora (de obicei din două litere), iar dacă s-a încărcat în prealabil şi setul de comenzi în limba română se pot introduce şi comenzi în limba română.

Toate formele au acelaşi efect, dar, evident este mai simplu să se folosească forma prescurtată (atît pentru limba engleză cît şi pentru română). La prezentarea comenzilor, vom oferi forma completă şi forma prescurtată (dacă există) în limba engleză, vom explica efectul comenzii, apoi vom prezenta şi comanda echivalentă împreună cu forma ei prescurtată (dacă există) în limba română.

Comenzile **LOGO** simbolizează o acţiune sau un lucru. Ele pot fi scrise mai multe pe acelaşi rînd de ecran, dar cu condiţia de a fi separate între ele cel puţin printr-un spaţiu (SPACE). Se formează, astfel,

o linie de comenzi multiple. O astfel de succesiune de comenzi se poate continua chiar pe mai multe rînduri ale ecranului. O comandă nu se execută decît dacă la terminarea ei (sau la terminarea înșiruirii mai multor comenzi între care se află) **se apasă tasta CR**; denumirea acestei taste este „carriage return", adică „întoarce carul!" — după manevra de dactilografiere care se execută la terminarea oricărui rînd (linii) scris.

O linie LOGO începe întotdeauna cu promptul „?" și se termină prin acționarea tastei CR; o linie LOGO se poate întinde pe mai multe rînduri de ecran.

Acționarea tastei **CR** semnifică terminarea liniei **LOGO,** dar, în același timp, produce și **lansarea în execuție** a acesteia; comenzile liniei sînt executate rînd pe rînd, pînă la ultima, iar după executarea acesteia pe ecran apare din nou promptul „?". Dacă vreunul din cuvintele din linia executată nu este recunoscut de broasca țestoasă ca fiind o comandă **LOGO** sau dacă comanda a fost scrisă greșit (atenție, deci, la ortografie, broasca este neiertătoare), execuția se întrerupe și apare în partea de jos a ecranului mesajul :

Nu știu cum să . . . .
(vezi fig. 2).
urmat de cuvîntul buclucaș.

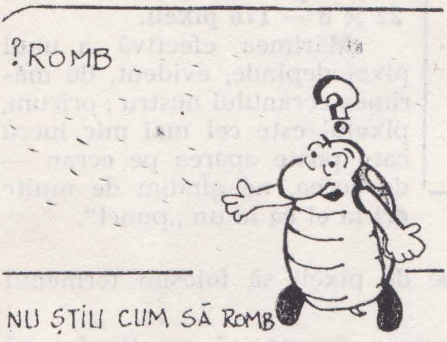


Fig. nr. 2

Deci, broasca ne atenționează prin **mesaje de eroare**: propoziții care dau informații referitoare la greșelile făcute în scopul corectării lor.

După apariția unui mesaj de eroare, apăsarea oricărei taste va provoca reîntoarcerea în modul de comandă („?"), urmînd ca eroarea să fie îndepărtată prin tastarea unei comenzi (sau șir de comenzi) diferite.

Și acum, în sfîrșit, să învățăm primele comenzi LOGO. Broasca țestoasă devine vizibilă dacă tastăm comanda : **SHOWTURTLE** sau prescurtat **ST,** adică „arată broasca!"; nu uitați ca după „textul" comenzii să apăsați tasta **CR!**

În limba română vom putea da comanda **BROASCA.** La terminarea comenzii de mai sus, în centrul ecranului apare „broasca țestoasă", sub forma unui mic triunghi; el seamănă mai degrabă cu un „vîrf de săgeată", dar noi am convenit să-l „vedem" ca pe o veritabilă țestoasă. Reținem faptul că vîrful cel mai ascuțit al triunghiului ne arată încotro este **orientată** (îndreptată) broasca.

Broasca poate fi făcută oricînd **invizibilă** cu comanda **HIDETURTLE** sau prescurtat **HT,** adică „ascunde broasca". În limba română putem tasta **FARĂBROASCĂ** sau prescurtat FB. Notăm că **ST** și **HT nu modifică** nici locul broaștei pe ecran, nici orientarea ei.

Inițial broasca se află în centrul ecranului (vom vedea mai departe că aici este „casa" ei) și este orientată spre nord (ca în **fig.** 3).

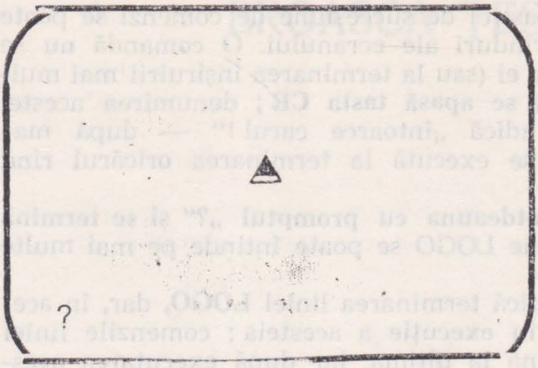


Fig. nr. 3

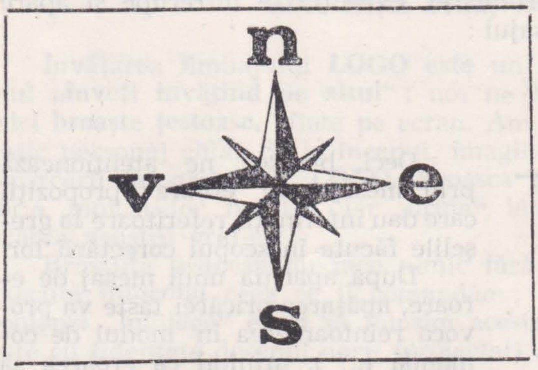


Fig. nr. 4

Convenția folosită este similară cu cea obișnuită de la busolă sau roza vînturilor, considerată orientare nord, partea de sus a ecranului **(fig. 4).**

Mişcările broaştei ţestoase

Partea utilă a ecranului are 22 de rînduri, fiecare rînd avînd 32 de poziții pentru caractere ; alte două rînduri de ecran, situate sub cele 22 amintite, sînt utilizate pentru afișarea comenzilor curente, dacă partea „principală“ este ocupată pentru operații grafice.

Spațiul rezervat pentru un caracter este un mic pătrat format din 8 × 8 = 64 **elemente de imagine** sau **pixeli ;** ecranul are, deci, pe orizontală 38×8= = **256** pixeli, iar pe verticală, 22 × 8 = **176 pixeli.**

Mărimea efectivă a unui pixel depinde, evident, de mărimea ecranului nostru ; oricum, pixelul este **cel mai mic** lucru care poate apărea pe ecran — de aceea, ne gîndim de multe ori la el ca la un „punct“.

Vom conveni, în **LOGO,** ca în loc de pixeli să folosim termenul de **pași.**

Înainte de a trece la realizarea unor desene, să menționăm că ecranul are două regimuri de funcționare :

— regimul **„textual“,** în care ecranul este utilizat pentru texte ;

— regimul **„grafic“,** în care ecranul este folosit pentru desene.

Inițial, ecranul este în regim textual, utilizînd toate cele 22 rînduri pentru texte ; la prima comandă grafică (de exemplu **BROASCA),** comenzi pentru **deplasarea** broaștei (înainte sau înapoi) și comenzilor curente fiind scrise de acum încolo pe cele două rînduri „suplimentare“.

Revenirea din regimul grafic în cel textual se face cu comanda : **TEXTSCREEN** (prescurtat **TS**), adică „ecran textual“.

Comenzile de mișcare a broaștei țestoase sînt de două feluri : comenzi pentru **deplasarea** broaștei (înainte sau înapoi) și comenzi pentru **rotirea** (schimbarea direcției) broaștei țestoase.

Pentru ca broasca să se deplaseze, ca urmare a unui ordin al nostru, ea trebuie să „știe“ cîți pași trebuie să facă ; ordinul respectiv trebuie să conțină, deci, pe lîngă cuvîntul „cheie“, și indicația cantitativă privind mărimea deplasării. O astfel de indicație valorică se mai numește **subiectul comenzii, parametru, parametru de intrare sau chiar intrare** și este separată de comanda propriu-zisă printr-un spațiu.

FORWARD 50 adică „mergi înainte 50 de pași !“, va avea ca efect deplasarea broaștei 50 de pași în direcția spre care a fost orientată **(fig. 3).** Același efect îl vor avea și forma prescurtată a comenzii, **FD 50,** precum și comenzile în limba română : **ÎNAINTE 50** sau **IN 50** (vezi fig. 5).

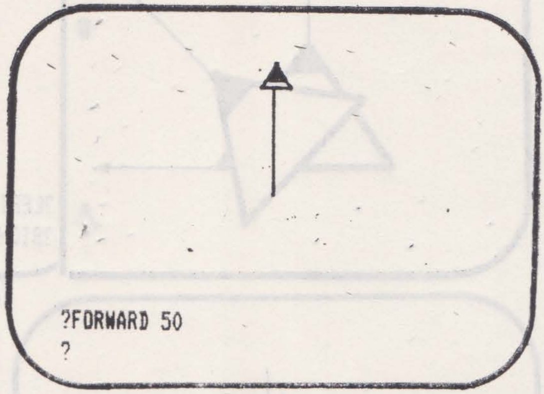


Fig. nr. 5

Pentru început vom comanda deplasări suficient de scurte pentru ca broasca să nu depășească marginile ecranului. Dacă, totuși, se va întîmpla o astfel de depășire, vom constata că broasca ieșită din ecran „reapare“ din partea opusă a acestuia ; vom vedea mai tîrziu cum pot fi modificate „efectele“ depășirii ecranului.



Fig. nr. 6

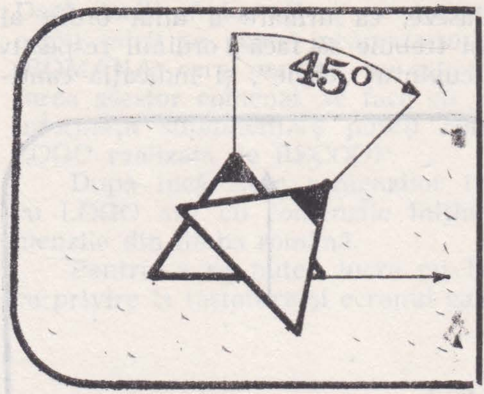
Este interesant să remarcăm de pe acum că, dacă parametrul de intrare al comenzii FORWARD este negativ (—5, —10, —20 etc.), atunci are loc o **retragere** a broaștei cu numărul respectiv de pași.

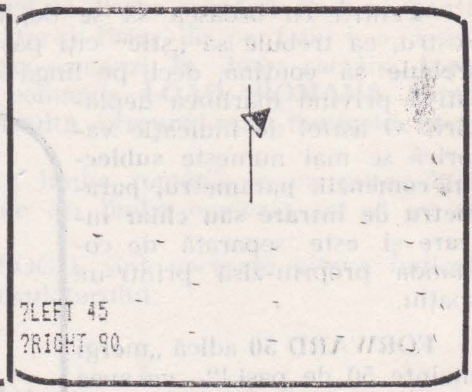
Dar există și o comandă specială pentru retragerea broaștei : **BACK 20** sau **BK 20,** adică „mergi înapoi 20 de pași“ (vezi **fig. 6).** Același efect îl va avea, bineînțeles, și comanda în limba română **INAPOI 20** sau **IP 20.**

Dacă la oricare din comenzile de deplasare a broaștei, se omite din greșeală specificarea valorii parametrului de intrare, calculatorul se oprește din execuție, după ce dă un mesaj de forma „prea puține intrări FD (sau BK)“.

La fel ca la comanda **FORWARD,** dacă parametrul de intrare al comenzii **BACK** este **negativ,** atunci are loc o **înaintare** a broaștei cu numărul respectiv de pași.

Rezultă, deci, că schimbarea semnului parametrului de intrare al comenzilor **FORWARD** și **BACK** inversează efectul acestora.





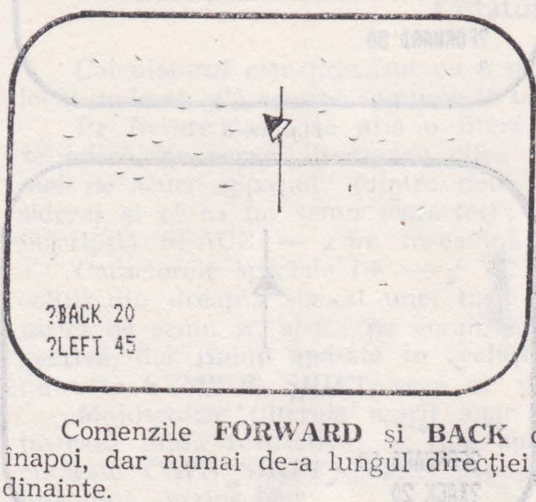


Fig. nr. 7

Fig. nr. 8

Fig. nr. 9

Comenzile **FORWARD** și **BACK** deplasează broasca înainte sau înapoi, dar numai de-a lungul direcției pe care broasca era orientată dinainte.

Schimbarea direcției de deplasare se poate realiza prin **rotirea** broaștei țestoase (la stînga sau la dreapta) cu un unghi corespunzător; mărimea unghiului se exprimă în grade (și eventual fracțiuni zecimale). Unghiul citat se măsoară de la verticală spre partea superioară și în sensul orar (vezi fig. 7). Comanda **LEFT 45** sau **LT 45** va însemna pentru broască „la stînga 45 de grade!" (vezi **fig. 8**). Similar în limba română avem **STÎNGA 45** sau **SA 45**.

RIGHT 90 sau **RT 90** va determina rotirea broaștei spre dreapta cu un unghi de 90 de grade. După această comandă (dată ulterior celei precedente) broasca va fi orientată pe o direcție spre dreapta, care face un unghi de 45 de grade cu axa verticală (vezi **fig. 9**).

Similar în limba română avem **DREAPTA 90** sau **DR 90.** Schimbarea semnului parametrului de intrare al comenzilor **LEFT** și **RIGHT** inversează efectele acestor comenzi; cu alte cuvinte, comanda **LT -60** este echivalentă cu comanda **RT 60.**

De remarcat că sensul rotirii (din comanda pe care o dați) se referă la „stînga" sau „dreapta" **broaștei țestoase.**

Trebuie remarcat caracterul diferit pe care îl au subiectele comenzilor pe care le-am studiat, cele de la comenzile **FORWARD** și **BACK** determină mărimea figurii, în timp ce pentru comenzile **RIGHT** și **LEFT** determină **forma** figurii.

Primele desene : Trasee și drumuri radiale

Pentru a „curăța" ecranul, vom da comanda : **CLEARSCREEN** sau prescurtat **CS**, adică „șterge ecranul". În limba română vom spune direct **ȘTERGE**.

Această comandă readuce broasca în poziție inițială, în centrul ecranului, redîndu-i și orientarea inițială. Readucerea acasă (în poziție și în orientare inițială) poate fi necesară și în unele cazuri în care nu dorim să dispară conținutul ecranului ; o astfel de revenire poate fi realizată cu comanda : **HOME**, adică „acasă" (în limba română **ACASĂ**).

Dar această comandă produce și trasarea pe ecran a drumului de revenire, ceea ce este de multe ori neconvenabil. Pentru a se înlătura acest inconvenient, se poate da în prealabil broaștei comanda de a **nu** trasa drumul pe care-l parcurge ; această comandă este :

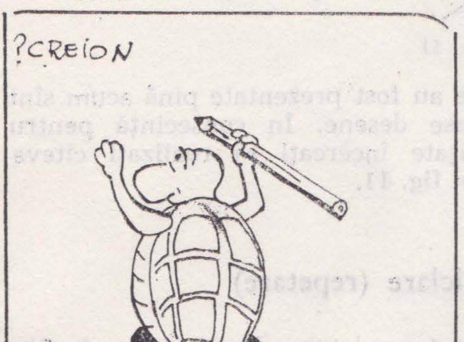






Fig. nr. 10 **CREION**

Fig. nr. 10 (a) **FĂRĂCREION**

Fig. nr. 10 (b) **GUMĂ**

PENUP sau prescurtat **PU,** adică „penița sus". În limba română vom spune că broasca este **FĂRĂCREION** sau prescurtat **FC.**

Din momentul în care primește această comandă, broasca țestoasă va rămîne cu penița ridicată (adică fără creion) pînă ce va primi comanda contrară : **PENDOWN,** sau prescurtat **PD,** adică „penița jos". În română se va da comanda **CREION** sau prescurtat **CR** (fig. 10). Astfel, revenirea „acasă", fără trasarea drumului de întoarcere și fără ștergerea ecranului se va realiza cu ajutorul succesiunii de comenzi : **PU HOME PD.**

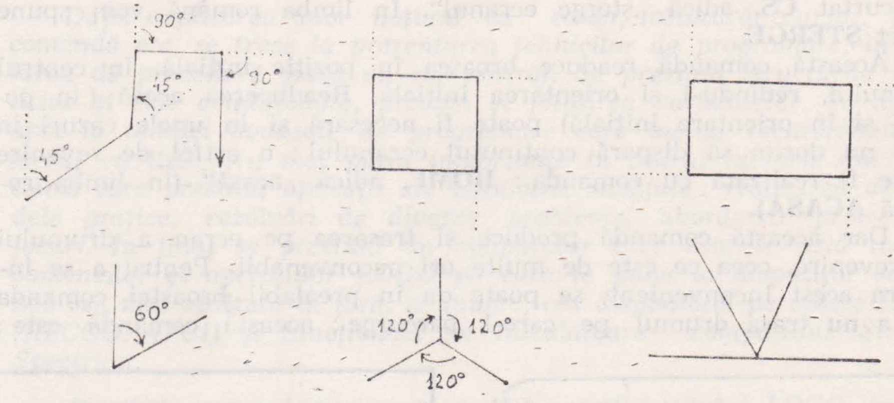


Fig. nr. 11

Elementele limbajului **LOGO,** care au fost prezentate pînă acum sînt suficiente pentru a realiza numeroase desene. În consecință pentru a vă familiariza cu comenzile învățate încercați să realizați cîteva desene geometrice, de exemplu cele din **fig. 11.**

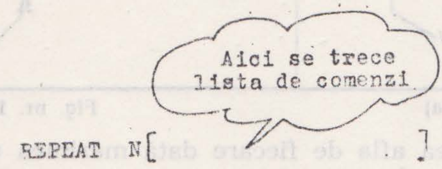
Instrucțiunea de ciclare (repetare)

Să încercăm să desenăm un pătrat cu latura de 60 de pași. Observăm că trebuie să repetăm de patru ori lista (șirul) de comenzi **FD 60 RT 90,** deoarece pătratul are 4 laturi și 4 unghiuri. De asemenea pentru a desena o „stea" formată din 3 raze de cîte 40 de pași dispuse astfel încît să formeze unghiuri egale, va trebui să repetăm de 3 ori lista de comenzi **FD 40 BK 40 RT 120.**

Situațiile în care trebuie să repetăm o listă de comenzi sînt foarte frecvente ; evident, este destul de incomod să înșirăm de multe ori același grup de comenzi — se pierde mult timp și este foarte plictisitor. Închipuiți-vă, de exemplu, ce s-ar întîmpla dacă am vrea să desenăm o stea cu ... 30 de raze ! Ar trebui să scriem de treizeci de ori același grup de comenzi : **FD 40 BK 40 RT 12 !**

Pentru a evita astfel de situații, limbajul **LOGO** conține o comandă de **repetare** a unui grup (șir) de comenzi ; deoarece repetarea se mai numește și „ciclare", această instrucțiune se mai numește **instrucțiune de ciclare.**

În mod natural, instrucţiunea de ciclare trebuie să aibe **două intrări** : una care să arate **de cîte ori** să se execute, iar alta care să arate **ce** să se execute ; prima intrare va fi, deci, un număr („N"), iar a doua intrare va fi o listă de comenzi. Instrucţiunea de ciclare prezintă numai forma completă :



adică „repetă de *n* ori comenzile din lista dată !". În limba română se foloseşte în loc de REPEAT cuvîntul **REPETĂ**. Lista de comenzi trebuie cuprinsă obligatoriu între paranteze drepte. Acestea se obţin cu tastele SS + Y şi respectiv, SS + U.

Cu ajutorul comenzii REPEAT, trasarea pătratului nostru se va putea face mult mai simplu cu linia LOGO :

REPEAT 4 [FD 60 RT 90]

Bineînţeles că dacă dorim să desenăm un pătrat cu latura mai mare sau mai mică, de exemplu 50, vom repeta linia LOGO, dar în loc de 60 vom tasta 50.

Iată şi liniile LOGO pentru desenarea altor figuri geometrice regulate (să convenim că dorim să le desenăm cu latura 30) (vezi fig. 12 şi fig. 13) :

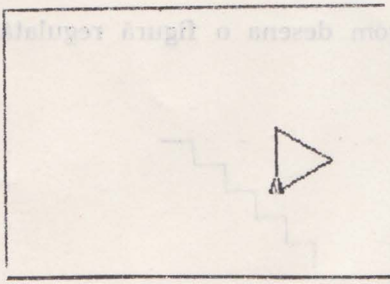


Fig. nr. 12

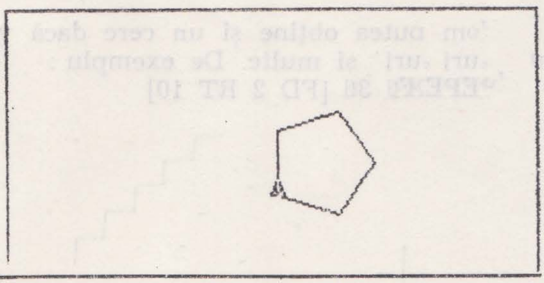


Fig. nr. 12 (a)

triunghi
REPEAT 3 [FD 30 RT 120]

pentagon
REPEAT 5 [FD 30 RT 72]

hexagon
REPEAT 6 [FD 30 RT 60]

octogon
REPEAT 8 [FD 30 RT 45]

După cum observaţi, regula este simplă : se repetă cele două comenzi referitoare la deplasare şi respectiv rotire de atîtea ori cîte laturi (sau unghiuri) are figura. Latura este tot timpul aceeaşi, deci subiectul lui **FD** rămîne acelaşi, modificîndu-se însă pentru fiecare figură subiectul lui RT.

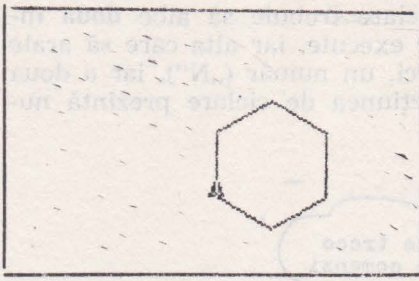


Fig. nr. 13 (a)

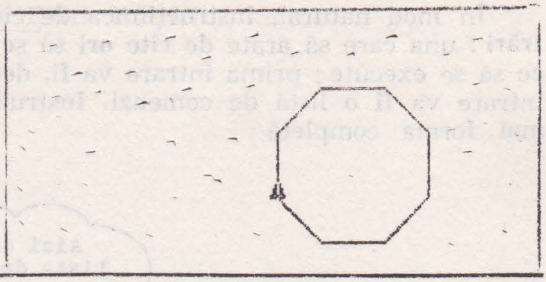
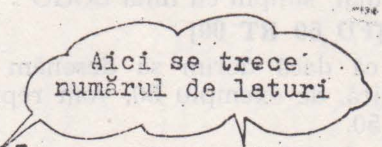


Fig. nr. 13 (b)

Cum se va putea afla de fiecare dată mărimea unghiului (este de fapt unghiul exterior deoarece broasca se rotește prin exteriorul figurii) ? Simplu ! Deoarece broasca ajunge după desenarea fiecărei figuri cu orientarea și în poziția în care era înainte de desenarea figurii, înseamnă că a efectuat o rotire completă (360°). Unghiul de rotire, deci, se va afla împărțind 360 la numărul de laturi pe care le are figura. Iată și formula generală pentru desenarea oricărei figuri regulate cu latura de 30 de pași :




```
REPEAT N [FD 3o RT 36o/N]
```

Exprimarea 360/N înseamnă 360 împărțit la N.

Vom putea obține și un cerc dacă vom desena o figură regulată cu laturi mici și multe. De exemplu :
REPEAT 36 [FD 2 RT 10]

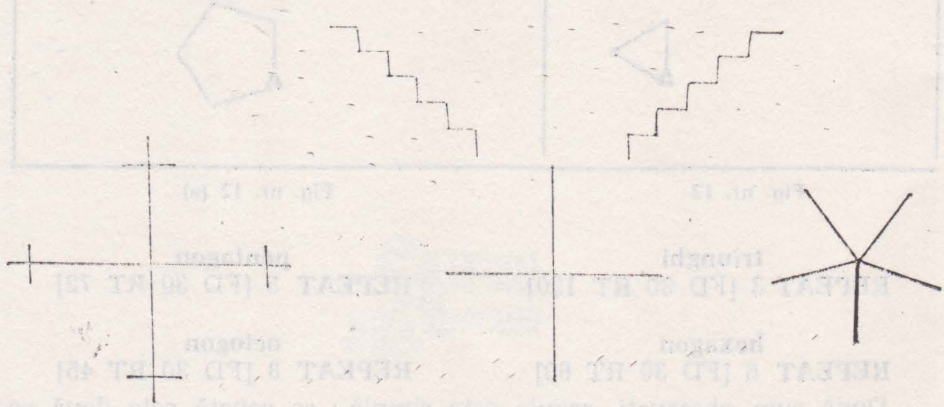


Fig. nr. 14

Încercați să realizați desenele din fig. 14, folosind instrucțiunea de ciclare. Care este formula generală pentru desenarea oricărei stele (cu orice număr de raze) ?

PROCEDURI

Ce este și cum se scrie o PROCEDURĂ

Instrucțiunea de ciclare (REPEAT) face să crească foarte mult randamentul muncii noastre cu calculatorul : prin puține instrucțiuni scrise, comandăm calculatorului să execute multe acțiuni. Vom vedea, în cele ce urmează, că există și alte posibilități de a crește acest randament, adică de a scădea numărul de comenzi necesare pentru ca broasca să execute o suită bogată de acțiuni.

Să considerăm, mai întîi, cazul unor desene relativ simple realizate de noi : pătratul, dreptunghiul, steaua ; după ce desenează oricare din aceste figuri, calculatorul (sau „broasca țestoasă") „uită" ceea ce tocmai a executat. Ori de cîte ori vrem să mai desenăm figura respectivă, trebuie să scriem din nou toate comenzile necesare.

Ar fi foarte bine dacă broasca ar putea „ține minte" cum se desenează un pătrat, dreptunghi etc., și ar executa desenul respectiv la simpla comandă PĂTRAT, DREPTUNGHI etc.

Ei bine, acest lucru este perfect posibil ! Limbajul LOGO ne oferă posibilitatea de a completa „bagajul de cunoștințe" al broaștei țestoase, permițîndu-ne să o învățăm sensul (înțelesul) unor cuvinte (comenzi) noi.

Să luăm, de exemplu, cazul unui pătrat cu latura de 30 ; pentru ca, la comanda PĂTRAT, broasca să deseneze imediat pătratul respectiv, trebuie să „știe" ce înseamnă A FACE un PĂTRAT. Altfel spus, ea trebuie să aibă în memorie înșiruirea (secvența) respectivă de comenzi ; această înșiruire îi arată cum se procedează cînd primește comanda cu numele respectiv.

Se numește *procedură* orice secvență de instrucțiuni memorate de calculator sub un *nume*, în vederea executării ei la întîlnirea numelui respectiv.

Introducerea procedurii în memorie se mai numește și definirea ei ; definirea procedurii are loc o singură dată, pe cînd executarea ei are loc ori de cîte ori se cere aceasta, prin indicarea numelui procedurii. Definirea procedurilor (introducerea lor în memorie) se face cel mai bine cu ajutorul Editorului LOGO ; lansarea editorului în acțiune se face cu comanda :

ED nume   sau   EDIT nume
unde „nume“ este numele procedurii pe care vrem să o definim. Avem toată libertatea în alegerea numelor de proceduri.

Să definim împreună procedura PĂTRAT, de care vorbeam mai sus ; tastăm comanda de apelare (lansare în acțiune) a editorului :
ED PĂTRAT

De pe ecran a dispărut promptul „?“ și a apărut „linia de titlu“ :
TO PĂTRAT

avînd cursorul pe litera T ; în partea de jos a ecranului apare scris „Editor LOGO“, ceea ce ne arată că toate comenzile pe care le scriem pe ecran vor fi preluate de editor, nefiind lansate acum în execuție.

Particula **TO** este folosită în limba engleză pentru a forma infinitivele verbelor, similar cu **A** din limba română (a fi, a face etc.) ; linia de titlu a oricărei proceduri începe cu această particulă tocmai pentru a arăta că textul care urmează este o procedură. Traducerea titlului ar fi, deci, „A (face un) PĂTRAT“.

Coborîm cursorul pe linia următoare (cu manevra CS + 6 și scriem comenzile necesare desenării pătratului :

REPEAT 4 [FD 30 RT 90]
(terminînd cu tasta (CR) !)

Comenzile formează, aici, o singură linie LOGO, dar, în cazul altor proceduri, vom avea mai multe linii care formează textul procedurii.

Terminarea procedurii trebuie să fie marcată printr-o linie LOGO specială, care conține doar cuvîntul **END** (sfîrșit) ; textul de pe ecran este, acum :

TO PĂTRAT

REPEAT 4 [FD 30 RT 90]

END
cursorul aflîndu-se pe linia următoare.

Deși noi am terminat de scris procedura, editorul LOGO nu și-a încheiat acțiunea ; dacă dorim, putem scrie în continuare și alte proceduri, fiecare cu linia ei de titlu, liniile cu comenzi și linia de sfîrșit corespunzătoare. Editorul depune în memoria calculatorului procedura (sau procedurile) scrise atunci cînd i se cere aceasta, prin manevra de „ieșire“, care are două etape :

— se trece tastatura în regim „E“ (CS + SS) ;

— se apasă tasta C

Efectuînd acum ieșirea din editor, observăm că ecranul se „curăță“ și, apoi, apare pe el mesajul :
PĂTRAT defined
care arată că procedura PĂTRAT a fost definită, adică a fost „depusă“ în memorie.

Cum verificăm că procedura noastră este bună ?

Simplu ! Dînd comanda de executarea ei, comandă identică tocmai cu numele procedurii :
PĂTRAT

Ecranul ne arată că procedura este corectă !

Desenaţi două pătrate noi pe ecran, lăsîndu-l acolo şi pe cel vechi ; folosiţi procedura definită de noi !

Ştergeţi ecranul ! Desenaţi un pătrat rotit spre stînga cu 45° ; desenaţi apoi unul rotit spre dreapta cu 30°.

Modificarea procedurilor

Cînd scriem textul unei proceduri putem comite, bineînţeles, diferite erori ; dacă le observăm înainte de „ieşirea din editor", soluţia este simplă — nu ieşim din editor pînă nu efectuăm corecturile necesare. Pentru aceasta, manevrăm cursorul aşa cum s-a arătat şi efectuăm ştergerile şi adăugirile de caractere după caz.

De multe ori, însă, erorile din textul unei proceduri devin „vizibile" abia cînd procedura se execută pentru prima oară şi constatăm că ea acţionează altfel decît am fi vrut noi. Rezultă că textul procedurii nu este corect şi, deci, trebuie modificat în mod corespunzător.

Modificările ulterioare ale unei proceduri se execută tot cu ajutorul Editorului. Apelarea (chemarea) editorului în vederea modificării unei proceduri definite anterior se face ca şi în cazul scrierii iniţiale a procedurii ; acum însă, deoarece procedura există în memoria calculatorului, textul ei va fi tipărit pe ecran în întregime, imediat după comanda noastră de apelare :

? ED nume procedură ....→ scris de noi
TO nume procedură }
text procedură } ....→ scris de către editor
END }

Avînd în faţă textul procedurii, putem face în el orice modificări dorim ; la terminarea modificărilor, se iese din editor prin manevra cunoscută (trecerea în regim „E", urmată de apăsarea tastei (C)).

Reamintim că fiecare linie LOGO scrisă se termină cu apăsarea tastei (CR) ; deşi la apăsarea acestei taste nu apare nimic pe ecran, în memoria calculatorului se introduce, în locul respectiv, un simbol de „sfîrşit de linie". Acest simbol trebuie tastat ca orice alt caracter ; el poate fi şters sau introdus după voia noastră.

Am menţionat acestea deoarece, de multe ori, cînd se fac modificări în proceduri, se comit greşeli legate de tasta (CR) : se apasă această tastă (în loc de SPACE) cînd cursorul se află în mijlocul unui rînd sau se apasă (CR) cînd cursorul se află chiar la începutul unui rînd. În primul caz, rîndul existent se „rupe" în două, iar în al doilea caz apare un rînd „gol".

Repararea acestor erori este simplă : trebuie şters „sfîrşitul de linie" (invizibil) care a fost introdus unde nu trebuia. Pentru aceasta, ducem cursorul pe primul caracter al liniei următoare şi facem manevra de ştergere de caracter — (CS) + (∅). În faţa cursorului aflîndu-se tocmai „sfîrşitul de linie" buclucaş, acesta este înlăturat şi situaţia revine la normal.

După scrierea unei proceduri, este obligatorie „lansarea" în execuție a procedurii respective pentru a verifica dacă ea „funcționează" așa cum vrem noi. Dacă procedura nu acționează cum trebuie, este necesar să identificăm greșelile din textul ei, să facem modificările corespunzătoare și să reluăm procesul (execuție, verificare, eventuale modificări) pînă cînd procedura este „pusă la punct". Acest proces se numește, de obicei, **testarea** procedurii.

Reținem că orice procedură trebuie să aibă structura din fig. 15.

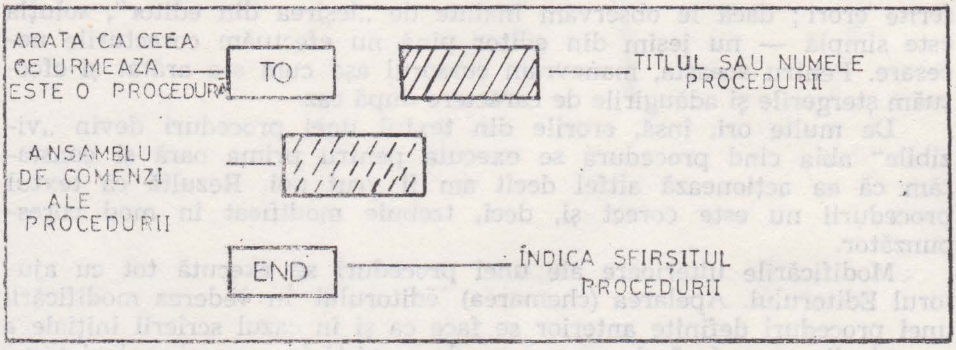


Fig. nr. 15

Modificați procedura PĂTRAT, astfel încît ea să deseneze pătrate cu latura de 20 de pixeli. Testați noua procedură.

Definiți procedura DREPT pentru desenarea unui dreptunghi de 30×50. Testați funcționarea ei.

Definiți procedura STEA pentru desenarea unei „stele" cu 20 de raze a cîte 15 pixeli fiecare (fig. 16). Testați procedura.

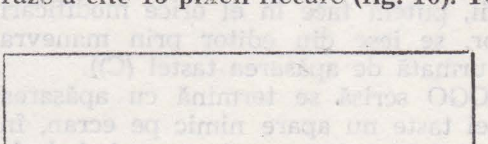


Fig. nr. 16

Supraproceduri și subproceduri

Numele unei proceduri definite de noi devine, după cum am văzut, o comandă nouă ; această comandă poate fi utilizată de noi la fel cum utilizăm comenzile LOGO „obișnuite", pe care broasca le știe „de la început". De altfel, comenzile LOGO se numesc tot „proceduri" însă, deoarece sînt cunoscute de broască „din primul moment", mai sînt numite **„proceduri primitive** sau, mai scurt, **primitive.** Exemple de primitive am întîlnit destule pînă acum : FD, RT, ST, PU, REPEAT etc.

Revenind la procedurile definite de utilizatori, vom remarca faptul că, putînd fi utilizate ca toate comenzile LOGO, **ele pot figura și în textele unor alte proceduri definite de** utilizatori. De exemplu, așa cum procedura PĂTRAT conține comenzile FD, RT etc., alte proceduri definite de noi vor putea conține, dacă este necesar, comanda PĂTRAT!

Spunem că o astfel de procedură **apelează** (cheamă) procedura PĂTRAT; ea este o **supraprocedură** pentru PĂTRAT iar PĂTRAT este o **subprocedură** pentru ea. Evident, orice procedură poate fi o subprocedură pentru unele proceduri și o supraprocedură pentru altele.

Să concretizăm cele de mai sus definind o procedură care să deseneze un „steag" cu pînza pătrată, avînd partea bățului aflată sub pînză egală cu latura pătratului; deoarece procedura PĂTRAT desenează pătrate cu latura de 20, procedura noastră (STEAG) va arăta astfel (definiți-o cu ajutorul Editorului):

```
TO STEAG
FD 20
PĂTRAT
BK 20
END
```

Am introdus comanda BK 20 pentru ca broasca să ajungă în final acolo unde se afla la început; vom respecta întotdeauna această „regulă" a cărei importanță o vom înțelege mai tîrziu.

Testați procedura STEAG.

Definiți și testați o procedură numită POM, care să deseneze un pom cu tulpina de 50, avînd drept „coroană" steaua desenată de procedura STEA (vezi fig. 17).

Definiți și testați o procedură (numită cum vreți) care să deseneze opt steaguri ce pornesc din același punct, fiecare lance („băț") formînd cu următorul un unghi de 45°.

Analog, o procedură pentru șase pomi ce pornesc din același punct și ale căror tulpini formează unghiuri de 60°.

Folosind procedura PĂTRAT, scrieți cîte o procedură pentru realizarea fiecărui din desenele figurii 18.

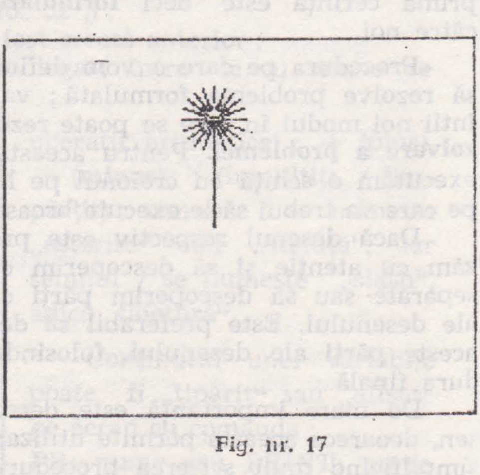


Fig. nr. 17

Indicație: cel mai simplu este să repetați de 4 ori desenarea unui pătrat, urmată de retragerea și rotirea convenabilă a broaștei. Pentru

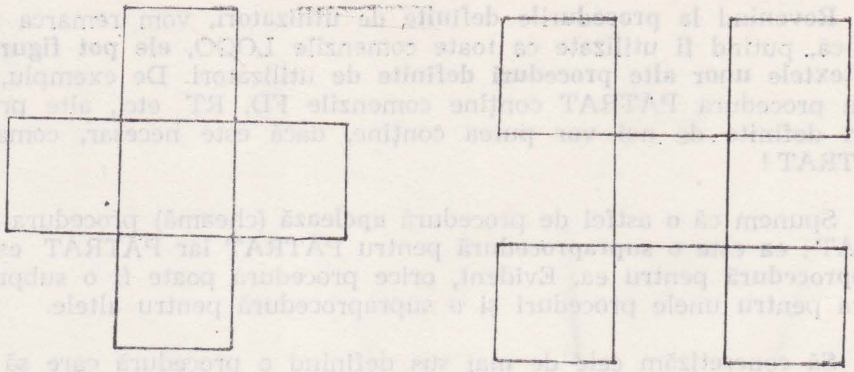


Fig. nr. 18

primul desen folosiți procedura PĂTRAT „pe dreapta", iar pentru al doilea desen modificați procedura PĂTRAT pentru desenarea pătratului „pe stînga".

Recomandări privind lucrul cu procedurile

Exercițiile precedente ne-au convins de faptul că definirea și utilizarea procedurilor este deosebit de fructuoasă, permițîndu-ne realizarea unor desene complicate cu un efort minim din partea noastră. Este necesar, însă, să ținem seama de cîteva principii care ne vor simplifica lucrul cu procedurile și ne vor ajuta să „stăpînim" cu ușurință proceduri din ce în ce mai complicate.

În primul rînd, vom porni de la ideea că o procedură ne va rezolva **o problemă** (la început „grafică", iar apoi și de alte tipuri); prima cerință este deci **formularea clară** a problemei respective de către noi.

Procedura pe care o vom defini „instruiește" broasca țestoasă **cum** să rezolve problema formulată; va trebui, deci, să ne clarificăm mai întîi noi modul în care se poate rezolva problema, **să găsim calea de rezolvare a problemei.** Pentru aceasta, de foarte multe ori este bine să executăm o schiță cu creionul pe hîrtie, gîndindu-ne la toate acțiunile pe care va trebui să le execute broasca, rînd pe rînd.

Dacă desenul respectiv este prea complicat, va trebui să-l analizăm cu atenție și să descoperim cum poate fi „descompus" în părți separate sau să descoperim părți care se repetă în mai multe locuri ale desenului. Este preferabil să definim proceduri mai simple pentru aceste părți ale desenului, folosindu-le apoi ca suproceduri în procedura finală.

De mare importanță este descoperirea „repetițiilor" dintr-un desen, deoarece aceasta permite utilizarea instrucțiunii de ciclare REPEAT, simplificînd mult scrierea procedurilor. Un exemplu de repetiție „ascunsă" l-am avut în cazul dreptunghiului, unde se repetă de două ori cuplul lățime-lungime.

Definiți și testați o procedură (numită TRI) pentru desenarea unui triunghi echilateral cu o latură orizontală (fig. 19).

Definiți și testați o procedură numită TRAP, pentru desenarea unui trapez ca acela din fig. 20.

Definiți și testați o procedură care să deseneze 20 de dreptunghiuri cu un vîrf comun, rotite fiecare față de cel precedent cu același unghi ; folosiți ca subproceduri pe DREPT.

Aceeași problemă pentru triunghiuri.

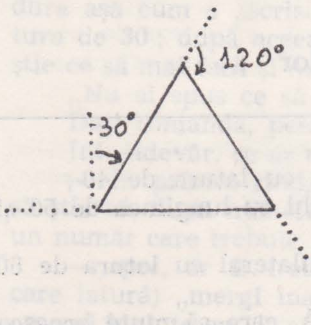


Fig. nr. 19

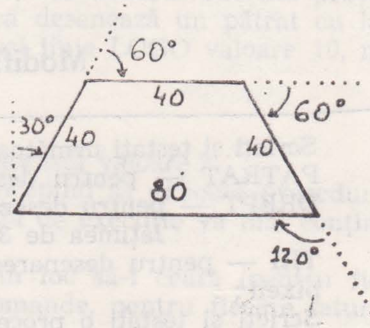


Fig. nr. 20

De multe ori se întîmplă să nu mai ținem minte numele tuturor procedurilor definite de noi ; afișarea titlurilor tuturor procedurilor se face la comanda : POTS (care înseamnă „Print Out Titles", adică „tipărește titlurile !").

Evident, înainte de a da această comandă este bine să trecem ecranul în regim textual, cu comanda TS.

O procedură devenită inutilă poate fi „ștearsă" din memorie cu comanda :
ER "nume" sau ERASE "nume"
adică „șterge procedura cu numele dat !".

Evident, această procedură trebuie să fie folosită cu mare atenție, numai cînd este strict necesar !

Afișați numele tuturor procedurilor create de voi !

Ștergeți procedura cu numele PĂTRAT. Dați comanda de desenare a steagului. Ce s-a întîmplat ? Redefiniți procedura PĂTRAT. Desenați steagul.

Dacă modificăm numele unei proceduri (cu ajutorul Editorului), vechea procedură nu se șterge din memorie ; în consecință, vor exista două proceduri identice sub două nume diferite. Evident, nu este util să creăm astfel de situații ; putem folosi însă această modificare de nume în cazul în care vrem să definim o procedură care diferă numai cu puțin de o procedură deja definită. Intrăm cu editorul în vechea procedură, facem modificările în text astfel încît să obținem textul noii proceduri și, în final modificăm și numele în mod corespunzător.

Pe baza procedurii PĂTRAT, definiți procedura PĂTRAT care să deseneze un pătrat „mare", cu latura de 50.

Modificarea dimensiunilor

Scrieţi şi testaţi următoarele proceduri :

PĂTRAT — pentru desenarea unui pătrat cu latura de 40 ;

DREPT — pentru desenarea unui dreptunghi cu lungimea de 50 şi lăţimea de 30 ;

TRI — pentru desenarea unui triunghi echilateral cu latura de 60 pixeli ;

Scrieţi şi testaţi o procedură numită MUTĂ, care să mute broasca ţestoasă pe orizontală la dreapta cu 70 de pixeli şi pe verticală în sus cu 30, dar **fără a trasa drumul** parcurs.

Ce facem dacă trebuie să desenăm un pătrat cu latura diferită de 40, cît prevede procedura noastră, PĂTRAT ?

Evident, vom modifica procedura existentă, punînd în textul ei, în loc de FD 40, deplasarea FD 30, FD 50, sau cît dorim. Procedura astfel modificată o putem „salva" sub un alt nume, să zicem PĂTRAT 2. Dacă dorim mai multe pătrate, de diferite dimensiuni, ar trebui, în acest fel, să îngrămădim în memorie mai multe proceduri, cu diferite nume.

Acest mod de lucru nu este convenabil nici pentru calculator, nici pentru noi, deoarece ocupă mult din memoria calculatorului şi ne obligă pe noi să ţinem minte ce face fiecare din procedurile respective.

O altă cale este aceea de a nu schimba numele procedurii, rămînînd ca ea să deseneze pătratul cu latura introdusă de noi la ultima utilizare.

Modificînd procedura PĂTRAT, dar fără a-i schimba numele, desenaţi cîte un pătrat cu latura de 20 şi respectiv 30.

În mod analog, desenaţi două dreptunghiuri cu dimensiunile de 70×20 şi respectiv 40×10.

Evident, nici acest procedeu de schimbare a dimensiunilor figurilor desenate nu este mulţumitor ; el ne obligă să apelăm de fiecare dată editorul şi, prin aceasta, să ştergem, de pe ecran figurile desenate anterior. Nu vom putea desena în acest mod niciodată două pătrate cu laturi diferite, care să fie în acelaşi timp pe ecran.

Proceduri cu parametri variabili

Numele procedurii fiind o comandă, ar trebui ca această comandă să fie astfel definită de noi încît să aibă nevoie de „intrări", adică de valori care să-i dirijeze acțiunea ; așa cum dăm comanda FD 10 sau FD 70, am vrea să putem da comanda PĂTRAT 10 sau PĂTRAT 70 și să se realizeze pătratul cu latura respectivă. Sau, în cazul desenării de dreptunghiuri, am vrea să putem da comenzi de genul DREPT 70 20, respectiv DREPT 40 10.

Dacă dăm acum comanda PĂTRAT 10, calculatorul execută procedura așa cum e „scrisă" în memorie, adică desenează un pătrat cu latura de 30 ; după aceea, intîlnind pe aceeași linie LOGO valoare 10, nu știe ce să mai facă și ne dă mesajul :

„Nu ai spus ce să fac cu 10".

Dați comanda, pentru a verifica.

Într-adevăr, ce ar trebui să facă el cu această valoare ?

— În primul rînd, ar fi trebuit ca, începînd să execute procedura, **să știe că,** după numele procedurii, comanda de execuție va mai conține un număr care trebuie „ținut minte" ;

— apoi, ar fi trebuit ca procedura, în loc să-i ceară (pentru fiecare latură) „mergi înainte cu 30", să-i comande, pentru fiecare latură, **„mergi înainte cu cît ai ținut minte".**

Iată, deci, că problema constă în a modifica procedurile în așa fel încît ele să poată utiliza memoria calculatorului pentru a depune aici anumite valori, pe care apoi să le poată regăsi și folosi.

Aducîndu-ne aminte că înseși procedurile sînt depuse în zone ale memoriei recunoscute prin **numele** procedurilor respective, ne dăm seama că și pentru memorarea diferitelor valori poate **fi folosit** același procedeu.

O „căsuță" (sau „locație") de memorie care a primit un nume dat de programator se numește **variabilă** ; această denumire se datorează faptului că locația respectivă poate conține, rînd pe rînd, diferite valori care se utilizează în același scop. Ca și în cazul procedurilor, programatorul nu trebuie să știe unde se află în memorie variabilele folosite de el ; aceasta este treaba calculatorului !

Mai apare o problemă : și variabilele și procedurile au „nume" după care sînt recunoscute ; dar cum știe calculatorul cînd ne referim la o procedură și cînd ne referim la o variabilă ?

Simplu ! **Limbajul LOGO ne obligă ca, atunci cînd ne referim la conținutul unei variabile, să punem semnul : (două puncte) înaintea numelui vairabilei respective.**

Să revenim la procedura PĂTRAT ; afișăm, cu editorul, textul ei :

```
TO PĂTRAT
REPEAT 4 [FD 30 RT 90]
END
```

Procedura trebuie modificată astfel încît să știe că, în momentul apelării (punerii în execuție), va primi o valoare pe care trebuie să o ia „în primire" și s-o depună într-o variabilă cu numele dat de noi ; în cazul nostru, vom numi cu **L** (de la „latură") variabila respectivă.

Pentru ca „rezervarea" căsuței (locației) corespunzătoare să se facă de la început, **Limbajul LOGO cere ca lista variabilelor care vor primi valori prin comanda de apelare să fie introdusă chiar în titlul procedurii, după numele acesteia.**

În cazul nostru, fiind vorba de o singură variabilă (L), vom completa linia de titlu astfel :
TO PĂTRAT : L

După ce, prin această modificare, am cerut să se rezerve o locație pentru variabila L, urmează să arătăm **cum** va folosi procedura conținutul acestei variabile. Evident, procedura trebuie să comande broaștei „înaintează cu : L" în loc de „înaintează cu 30", pentru fiecare latură a pătratului.

Coborîm, deci, cursorul, pe linia următoare, ștergem pe 30 de după FD și introducem în locul lui pe : L. Ieșim din editor și cerem desenarea pe rînd a pătratelor cu laturi de 10, 20, 70 etc. prin comenzile PĂTRAT 10, PĂTRAT 20, PĂTRAT 70 etc.

Desenați pe ecran trei pătrate de dimensiuni diferite.

Notînd cu LUN și LAT lungimea și lățimea dreptunghiului, modificați procedura DREPT, astfel încît ea să deseneze orice dreptunghi.

Notînd cu L latura triunghiului echilateral, modificați procedura TRI în mod analog.

Modificați procedura MUTĂ, astfel încît ea să mute broasca pe orizontală cu DO (deplasarea orizontală) și pe verticală cu DV (deplasare verticală).

Apelați procedura MUTĂ prin MUTĂ — 100 — 50.

Ce se întîmplă ? De ce ?

Studiați ce se întîmplă cu TRI și DREPT dacă „intrările" sînt negative.

Evident, de acum înainte nu vom mai scrie proceduri pentru dimensiuni „fixe" ale figurilor, ci ne vom gîndi de la început să stabilim care sînt mărimile ce determină (stabilesc) dimensiunile figurii dorite ; vom stabili, apoi, numele variabilelor respective și vom scrie procedura folosind aceste variabile. Procedura, astfel scrisă, va desena figura dorită la orice „scară", adică oricît de mare o dorim. Dacă am fi cunoscut de la început noțiunea de variabilă, am fi scris, bineînțeles, toate procedurile de pînă acum utilizînd variabile și nu ar mai fi fost necesar să le modificăm.

Variabile locale și variabile globale

Variabilele care apar ca parametri de intrare într-o procedură sînt create de procedură în momentul apelării și folosite în timpul executării procedurii respective ; la terminarea procedurii, aceste variabile sînt „distruse", deci nu mai putem utiliza conținutul lor. Din acest motiv, variabilele care sînt declarate ca parametri de intrare în proceduri se mai numesc și **variabile locale.**

Deoarece variabilele locale sînt create de fiecare procedură, se poate utiliza același nume de variabilă locală în diferite proceduri, fără ca prin aceasta să se creeze confuzii. De exemplu, putem folosi numele LAT pentru latura pătratului, ca și pentru latura triunghiului, ca parametru de intrare declarat în titlul ambelor proceduri ; fiecare procedură rezervă și utilizează variabila sa locală cu numele respectiv.

Totuși, sînt situații cînd am vrea ca o aceeași variabilă să poată fi utilizată de mai multe proceduri independente sau chiar de unele comenzi LOGO din afara procedurilor. O astfel de variabilă ar fi, deci, **globală.**

Variabilele globale pot fi create în proceduri sau în afara lor cu ajutorul instrucțiunii MAKE — „fă-l pe . . ." ; de ex. :
MAKE "A 50 sau MAKE "A 10 sau MAKE "BAU: A+3

Instrucțiunea MAKE creează variabila cu numele respectiv și depune la adresa corespunzătoare (din memorie) ceea ce urmează după numele variabilei (fig. 21) ; deoarece este vorba de „adresa de destinație" a unei valori, aici înaintea numelui variabilei se pun ghilimele ("), nu (:). Forma generală a instrucțiunii MAKE este :

MAKE "nume expresie

unde numele este al variabilei, iar ca expresii putem pune, deocamdată :

— un număr întreg ;
— un număr zecimal (scris cu . în loc de ,) ;
— conținutul unei variabile care a fost creată anterior ;
— mai multe numere sau variabile legate între ele cu semne de operații aritmetice : + (plus), — (minus), × (înmulțit), / (împărțit) ; semnul * se numește „asterisc" sau „steluță", iar semnul / se numește „slash", adică „tăietură".

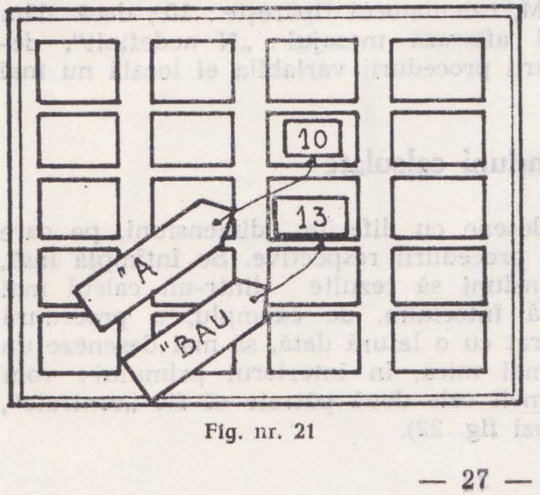


Fig. nr. 21

Conținutul unei variabile poate fi „tipărit" sau „afișat" pe ecran cu comanda :
PR: nume sau PRINT: nume adică „tipărește . ."

Treceți ecranul în regim textual!

Tastați următoarele linii LOGO și explicați-le înainte de a apăsa tasta CR:

```
MAKE "A 50 PR :A
MAKE "ION 3.5 * 3 PR :ION
PR 10/3
CS MAKE R :A + :ION — 0.5 PR :ION PĂTRAT :R
CS MAKE "ANA :"R/4 PĂTRAT :ANA
```

Instrucțiunea MAKE poate fi utilizată și pentru **modificarea** conținutului unei variabile (globale sau locale). Dacă dați acum comenzile MAKE "A 30 PR :A, calculatorul șterge vechiul conținut al variabilei A și îl înlocuiește cu 30, după cum vedem și din ce a scris el pe ecran.

În cazul modificării, poate fi folosită (în acea instrucțiune) vechea valoare a variabilei; de exemplu, dacă vrem să adăugăm ceva la valoarea unei variabile, scriem:

MAKE "A :A + 1 sau MAKE "A :A + 15 sau
MAKE "A :A + 0.75

Iată o procedură care pune calculatorul să „numere" de la 1 la un număr oarecare N:

```
TO NUMĂRĂ :N
TS MAKE "NUM 0
REPEAT :N [MAKE "NUM :NUM+1 PR : NUM]
END
```

Tastați procedura și explicați funcționarea ei!

De ce a fost necesară instrucțiunea MAKE "NUM 0?

Testați procedura pentru N=10 și N=100!

Remarcați faptul că variabila N este locală, fiind în lista de intrări a procedurii; variabila NUM, însă este o variabilă globală. Dacă dăm acum comanda: PR :NUM, calculatorul tipărește: 10; dacă dăm comanda: PR :N, calculatorul afișează mesajul: „N nedefinit", deoarece după încheierea executării procedurii variabila ei locală nu mai este utilizabilă.

Dimensiuni calculate

Pînă acum am executat desene cu diferite dimensiuni, pe care le-am indicat, de fiecare dată, procedurii respective. Se întîmplă însă, de multe ori, ca aceste dimensiuni să rezulte dintr-un calcul mai simplu sau mai complicat. Să întocmim, de exemplu, o procedură care după ce desenează un pătrat cu o latură dată, să mai deseneze un pătrat, cu tura de trei ori mai mică, în interiorul primului; vom „aranja" în așa fel lucrurile, încît cele două pătrate să fie „centrate", adică să aibă același centru (vezi fig. 22).

Primul pas va fi evident trasarea pătratului „mare“ (să numim latura lui cu LM) ; apoi vom muta broasca spre dreapta și în sus cu a treia parte din latura dată. Desenăm pătratul „mic“ și în sfîrșit vom muta broasca la loc, în punctul de unde a început desenul :

```
TO PĂTRATE :LM
PATRAT :LM
MUTĂ :LM/3 :LM/3
PĂTRAT :LM/3
MUTĂ —1 * :LM/3 —1 * :LM
END
```

Înmulțirea cu —1 s-a efectuat pentru a schimba semnele cantităților respective, în vederea mutării broaștei la stînga și în jos.

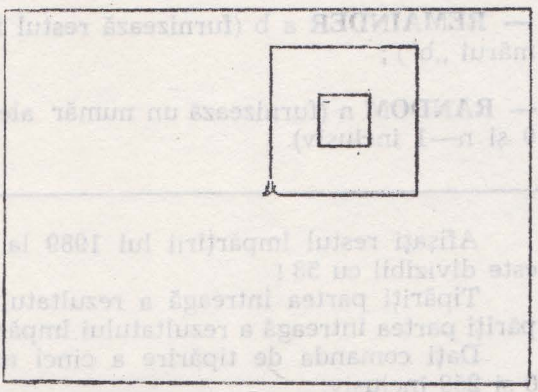


Fig. nr. 22

Scrieți și testați o procedură numită STEA, care să deseneze o stea cu N raze de lungime L ; în procedură veți calcula mai intîi și veți depune în variabila U mărimea unghiului format de fiecare rază cu următoarea.

Funcții (operații)

Pînă acum am numit „comenzi“ toate instrucțiunile LOGO ; este momentul să facem o distincție între două tipuri de instrucțiuni :

— se numesc **operații** sau **funcții** instrucțiunile a căror executare conduce la obținerea unei „valori“ ;

— se numesc **comenzi** instrucțiunile care nu sînt operații (funcții).

Primele operații pe care le-am întîlnit au fost chiar operațiile aritmetice (+, —, ×, /) ; în urma executării fiecăreia din acestea se obține o valoare numerică. Celelalte funcții (operații) LOGO nu se notează cu simboluri ci au nume, ca orice primitivă ; vom da ceva mai jos cîteva exemple.

Rezultatul unei operații LOGO trebuie să fie utilizat întotdeauna ca intrare pentru o comandă sau pentru o altă operație ; în caz contrar, calculatorul dă mesajul :

„nu ai spus ce să fac cu . . . .“

urmat de rezultatul cu care nu știe ce să facă.

De exemplu, operațiile aritmetice au fost utilizate de noi ca intrări pentru MAKE, pentru PR sau chiar pentru procedurile noastre (PĂTRAT, MUTĂ). Dacă tastăm 50/2+10 urmat de (CR), va apare mesajul :

„nu ai spus ce să fac cu 35“ !

În afară de operațiile limbajului LOGO are și alte funcții aritmetice; dintre acestea, amintim deocamdată doar cîteva :

— **INT** a (furnizează partea întreagă a numărului „a" înlăturînd zecimalele acestuia) ;

— **REMAINDER** a b (furnizează restul împărțirii numărului „a" la numărul „b") ;

— **RANDOM** n (furnizează un număr aleator (întîmplător) cuprins între 0 și n—1 inclusiv).

Afișați restul împărțirii lui 1989 la 17 ; vedeți dacă numărul 1999 este divizibil cu 53 !

Tipăriți partea întreagă a rezultatului împărțirii lui 256 la 45 ; tipăriți partea întreagă a rezultatului împărțirii lui 175 la 45.

Dați comanda de tipărire a cinci numere aleatoare cuprinse între 0 și 259 inclusiv.

Scrieți procedurile necesare pentru „umplerea" ecranului cu pătrate avînd aceeași latură (L) și avînd un spațiu de 5 pixeli între două pătrate alăturate.

Indicație. Procedura principală va trebui să calculeze numărul de rînduri (NR) și numărul de pătrate de pe fiecare rînd (NP).

Scrieți o procedură (numită TRAP) care să deseneze un trapez cu baza mare L, celelalte laturi fiind L/2, iar unghiurile ascuțite de 60°.

Scrieți o procedură care să deseneze un triunghi echilateral ; numiți-o TRI.

ACȚIUNI CONDIȚIONATE

O compoziție grafică

Ne propunem să desenăm o casă !

Corpul casei este bineînțeles un dreptunghi ; să notăm cu LU lungimea corpului casei și cu IN înălțimea acestui corp.

Casa noastră va trebui să aibe o ușă ; de asemenea, îi vom face și o fereastră. Dimensiunile acestora vor depinde, desigur, de dimensiunile casei (LU și IN) și deci vor trebui să fie calculate în funcție de ele (vezi fig. 23).

Să ne ocupăm mai întîi de dimensiunile orizontale : o schiță făcută pe tablă sau pe o hîrtie ne arată că fațada casei este împărțită în cinci părți astfel : interval — ușa — interval — fereastră — interval. Evident, cel mai simplu este ca toate acestea să aibe aceeași mărime, deci mărimea fiecăreia va fi, pe orizontală, LU/5.

Pe verticală în dreptul ferestrei avem trei părți : intervalul de jos, intervalul de sus ; le vom lua și pe acestea egale, deci fiecare va avea fereastra și dimensiunea verticală de IN/3. Cît privește ușa, pornind de jos și terminîndu-se la același nivel cu fereastra, va avea dimensiunea verticală dublă (2 * : IN/3).

Acoperișul trebuie să aibe baza ceva mai mare decît lungimea casei ; vom conveni ca acoperișul să „iasă în afară", în fiecare parte a casei, cu a zecea parte din lungimea ei (LU/10). Latura de bază a acoperișului va fi deci egală cu : LU + 2 * : LU/10.

Recapitulînd, procedura de desenare a casei va trebui să execute următoarele acțiuni pentru a realiza corpul casei :

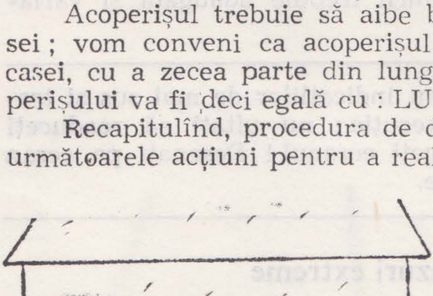


Fig. nr. 23

— mutarea broaștei la colțul ușii cu (LU/5 la dreapta, pe orizontală și cu 0 pe verticală) ;

— desenarea ușii (dreptunghi de dimensiuni LU/5 și 2 * IN/3) ;

— mutarea broaștei la colțul ferestei ;

— desenarea ferestrei ;

— readucerea broaștei în colțul casei ;
— mutarea broaștei în colțul de jos al acoperișului.

Folosind indicațiile de mai sus, scrieți procedura CASA (mai precis, partea care desenează corpul casei).

Instrucţiunea IF

Am lăsat la urmă problema acoperișului, deoarece aici părerile sînt de obicei împărțite, unora le place mai mult un acoperiș triunghiular (țuguiat), iar alții preferă un acoperiș de formă trapezoidală.

Noi avem pregătite procedurile respective (TRI și TRAP) ; problema pe care o avem de rezolvat este ca în funcție de dorința utilizatorului procedura CASĂ să apeleze cînd pe TRI, cînd pe TRAP.

Să numim casele cu acoperiș triunghiular „case de tipul 1", iar pe cele cu acoperiș trapezoidal „case de tipul 2". Procedura CASĂ ar trebui deci să primească și intrarea 1 sau 2, care să indice tipul casei dorite ; să numim TIP variabila locală respectivă care trebuie să fie adăugată în titlul procedurii.

Avînd cunoscut tipul casei, acțiunea de desenare a acoperișului va trebui să se desfășoare astfel :

Dacă : TIP =1, desenează un triunghi (cu TRI) ;
altfel, desenează un trapez (cu TRAP).

Instrucțiunea care comandă **executarea condiționată** a unor acțiuni se numește **IF** („dacă") ; ea are forma generală :

IF condiție [lista 1 de instrucțiuni] [lista 2 de instrucțiuni]

La intîlnirea acestei instrucțiuni, calculatorul verifică dacă în momentul respectiv condiția este îndeplinită ; dacă **da**, el execută lista 1 de instrucțiuni, iar în caz contrar execută lista 2 de instrucțiuni. Evident, el nu execută niciodată ambele liste (ori una ori alta !).

Procedura CASĂ va trebui deci să continue cu instrucțiunea :

IF :TIP=1 TRI :LU + 2 * :LU/10 TRAP :LU + 2 * :LU/10

Evident, în linia de titlu a procedurii trebuie adăugată și variabila : TIP.

Completați procedura CASĂ conform indicațiilor de mai sus și testați-o pentru ambele variante de execuție ; nu uitați să readuceți broasca în colțul inițial al casei ! Ștergeți ecranul ! Desenați pe ecran cinci case de dimensiuni și tipuri diferite.

Excluderea unor cazuri extreme

Din păcate casa noastră nu este întotdeauna la fel de frumoasă !

Desenați case de tipul 1, dimensiuni foarte mici : 2 și 1, 3 și 2, 4 și 3, 5 și 4, 6 și 4, 8 și 6 etc.

Observăm că dacă lungimea casei este mai mică decît 6, desenul nu seamănă aproape deloc cu o casă ; abia de la LU=6 poate fi recunoscută casa în desen.

Apare deci necesitatea de a se exclude cazurile în care desenul este diform, necorespunzător ; cum s-ar putea face aceasta ?

Procedura CASĂ ar trebui să includă, chiar de la început, o instrucțiune care să însemne :

Dacă :LU este mai mic decît 6 **oprește-te !**

Evident, va trebui să folosim un IF ; limbajul LOGO are și comanda de „oprire" a execuției unei proceduri înainte de epuizarea instrucțiunilor : STOP.

Vom introduce deci la începutul procedurii CASĂ, chiar după linia de titlu, linia :

IF :LU < 6 [STOP]

În consecință, dacă valoarea primită pentru lungimea casei este mai mică decît 6, procedura se oprește din start, adică nu desenează nimic și pe ecran apare promptul „?". Dacă valoarea respectivă este mai mare sau egală cu 6, procedura **nu se oprește** și trecînd la liniile următoare ne desenează casa.

Efectuați modificarea indicată în procedura CASĂ și testați procedura cu valori mai mici și mai mari ale lungimii.

Observăm că aici am utilizat o formă „incompletă" a lui IF :

IF condiție [lista de instrucțiuni]

Această formă cere executarea listei de instrucțiuni **dacă este îndeplinită condiția dată ;** în caz contrar lista nu se execută și se trece la linia LOGO care urmează.

Scrieți o procedură care, primind două intrări, s-o depună pe cea mai mare în variabila globală MAX, iar pe cea mai mică în variabila globală MIN. Scrieți o procedură care să calculeze valoarea absolută (modulul) unui număr dat. Reamintim că modulul unui număr este egal chiar cu acest număr dacă el este pozitiv sau zero și este egal cu opusul numărului, dacă acesta este negativ ; opusul se obține prin înmulțirea numărului cu —1.

Dacă o procedură se termină prin obținerea unei valori, acea procedură se aseamănă cu operațiile (funcțiile) LOGO ; pentru ca ea să se comporte întocmai ca aceste funcții, este necesar ca terminarea ei să marcheze (să accentueze) rolul de funcție pe care-l îndeplinește.

O procedură se transformă în „funcție" dacă se termină cu instrucțiunea de „ieșire" ; aceasta este :

OP expresie sau OUTPUT expresie

Prin urmare, procedura de aflare a modulului se poate scrie astfel :

```
TO VABS :A                  sau   TO VABS :A
IF :A > 0 [OP :A]                  IF :A < 0 [OP — 1 * :A] [OP :A]
IF :A=0 [OP :A]                    END
IF :A < 0 [OP — 1 * :A]
END
```

După cum se știe, funcțiile trebuie să apară întotdeauna ca intrări ale altor comenzi sau funcții ; și procedura VABS se va apela în aceleași condiții. De exemplu :

PR VABS —7 PR VABS —7 * 5 etc...

Scrieți o procedură pentru desenarea unui dreptunghi așezat orizontal, indiferent de ordinea intrărilor !

RECURSIVITATE

Apelarea recursivă

Ne-am obișnuit cu apelarea procedurilor de către alte proceduri; de exemplu, procedura CASĂ apelează patru proceduri: DREPT, MUTĂ, TRI și TRAP. Dar și înainte de această procedură am întîlnit astfel de exemple, la desenarea unui steag, a unui pom etc.

Limbajul LOGO permite chiar și apelarea unei proceduri ... **de către ea însăși!** Această apelare sau, mai bine zis, auto-apelare a unei proceduri se numește **apelare recursivă**. La prima vedere această idee poate părea ciudată și nu prea este clar la ce anume ar putea folosi „apelarea recursivă".

Exemplul următor ne va ajuta să ne facem o primă imagine asupra mecanismului apelării recursive și, totodată, să înțelegem utilitatea acestui procedeu.

Să încercăm să exemplificăm funcționarea procedurii următoare, numită

MODEPAT (MOdel DEcorativ PĂtrat):

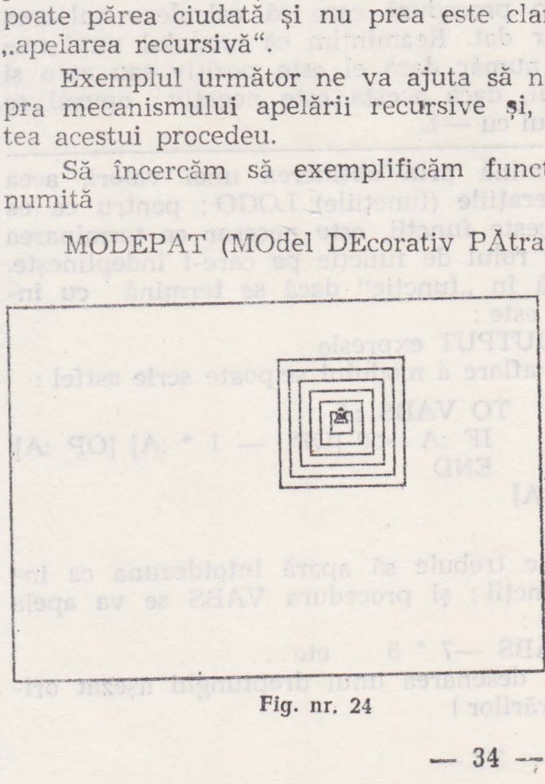


Fig. nr. 24

```
TO MODEPAT :L :P
PĂTRAT :L
MUTĂ :P :P
MAKE "L :L — 2* :P
IF :L > 0 [MODEPAT :L :P]
END
```

Testați procedura pentru L=60 și P=5! (vezi fig. 24).

Explicați acțiunea procedurii MODEPAT și schițați pe hîrtie desenul care urmează să fie trasat de această procedură.

Introduceți procedura în memorie și testați-o. Ce reprezintă L ? Dar P ?

Să recapitulăm deci acțiunile întreprinse de procedură :

— Desenează un pătrat cu latura L ;

— Se deplasează la dreapta și în sus cu P (și P) ;

— Modifică valoarea laturii L, micșorînd-o cu 2*P ;

— Dacă L este încă mai mare decît 0, se execută din nou toate instrucțiunile, începînd cu primul punct.

S-a văzut că modificarea laturii se face de fiecare dată în așa fel încît pătratul următor să fie cuprins în cel dinainte ; este important să subliniem că micșorarea de fiecare dată a laturii face ca procesul reluării să aibe, în mod sigur, un sfîrșit. Într-adevăr deși **nu se știe dinainte cîte repetiții vor avea loc**, tot micșorîndu-se, „odată și odată" latura va trebui să devină egală sau chiar mai mică decît zero. În acel moment, condiția din IF nu mai este îndeplinită și, deci, procedura nu se mai reia prin auto-apelare.

Una din observațiile de mai sus merită mai multă atenție : procedura noastră, deși repetă de mai multe ori executarea acelorași instrucțiuni, **nu „știe" de la început de cîte ori trebuie să repete execuția.** Iată de ce desenul nostru nu a putut fi realizat prin utilizarea unui REPEAT.

Prin urmare, **apelarea recursivă a unei proceduri trebuie folosită ori de cîte ori este necesară executarea repetată a unor instrucțiuni, dar nu se poate afla dinainte numărul de repetări.**

Dar, pentru a nu se produce o repetare fără sfîrșit a acțiunilor procedurii, **apelarea recursivă trebuie să fie întotdeauna condiționată,**

Un exemplu de repetare fără sfîrșit îl putem avea chiar și cu procedura noastră, dacă luăm pasul P egal cu 0 ; într-adevăr, în acest caz mutarea broaștei spre interior nu se produce, latura nu se micșorează, deci condiția de repetare este tot timpul îndeplinită. Dacă am făcut greșeala să apelăm procedura cu valoarea 0 și pentru P, va trebui să întrerupem execuția cu manevra CS + SPACE.

Dați comanda MODEPAT 70 0 ; ce se întîmplă ?

Întrerupeți execuția ; dați comanda PD, pentru ca, dacă întreruperea s-a produs în subprocedura MUTĂ, broasca să nu rămînă cumva cu penița sus.

Contorizare și însumare

Am utilizat apelarea recursivă deoarece nu știm de cîte ori va trebui mutată broasca țestoasă ; ne punem acum problema ca, măcar la terminarea procedurii, să știm cîte mutări „de la colț la colț" a efectuat broasca. Evident, va trebui să **numărăm** aceste mutări, deci vom modifica în mod corespunzător procedura noastră. Să o afișăm cu editorul :

```
TO MODEPAT :L :P
          ←  MAKE "K 0
PĂTRAT :L
MUTĂ :P :P
          ←  MAKE "K :K + 1
MAKE "L :L — 2 * :P
IF :L > 0 MODEPAT :L :P
END
```

Aducîndu-ne aminte cum am realizat o „numărătoare“ cu calculatorul, ne gîndim că, pentru a număra mutările efectuate de broască, ar trebui să dăm valoarea inițială 0 unui contor numit, de exemplu, K; apoi ar trebui să adăugăm valoarea 1 acestui contor, de fiecare dată cînd se mută broasca. Vom spune că prima instrucțiune **inițializează** contorul iar a doua **avansează** contorul. Săgețile de mai sus indică locurile unde ar trebui introduse aceste instrucțiuni.

Din păcate, procesul de numărare pe care-l încercăm astfel **nu poate funcționa**; într-adevăr, în cazul nostru repetarea acțiunilor are loc prin auto-apelare (apelare recursivă). În consecință, fiecare apelare a procedurii de către ea însăși, prin ultima instrucțiune, face ca inițializarea contorului să se efectueze din nou! Variabila K va lua, succesiv, valorile 0, 1, 0, 1, 0, ... etc.

Evident, inițializarea variabilei K trebuie să se facă în afara procedurii MODEPAT; avansul contorului însă trebuie să rămînă unde l-am introdus. Vom scrie deci o „supraprocedură“ (s-o numim MODEP), care va efectua următoarele acțiuni:

— inițializarea contorului;
— apelarea procedurii MODEPAT;
— aducerea broaștei în colțul inițial al desenului.

Acest ultim punct l-am introdus ca să „profităm“ de numărătoarea atît de „greu“ realizată; de altfel, aducerea broaștei în poziția inițială a fost scopul nostru nemărturisit.

Introduceți instrucțiunea de avans al contorului în MODEPAT.

Scrieți procedura MODEP, care să realizeze cele trei acțiuni de mai sus; țineți seama că s-au efectuat K mutări de mărime P, atît pe orizontală, cît și pe verticală.

Dacă în variabila K am fi adunat direct cu cîți pași s-a deplasat broasca la fiecare mutare, în final variabila K ne-ar fi indicat direct deplasarea totală a broaștei, iar instrucțiunea de readucere a ei „la loc“ ar fi fost mai simplă.

Modificați cele două **proceduri** în felul indicat de această observație.

Cele de mai sus ne permit să ne facem o imagine mai clară asupra **procesului de adunare** a unui șir finit de numere; pentru început,

trebuie să inițializăm cu 0 o variabilă care în final va conține suma. Apoi trebuie parcurs șirul numerelor respective, adunînd de fiecare dată numărul la variabila-sumă.

Vom concretiza cu șirul numerelor naturale : să facem suma numerelor naturale mai mici sau egale cu N ! Pentru aceasta :

— inițializăm cu 0 variabila S (suma) ;
— inițializăm cu 0 variabila NC (numărul curent) ;
— repetăm de N ori :
  — adunarea lui „1" la NC (aflarea numărul „actual")
  — adunarea lui NC la S ;
— tipărim rezultatul final (S).

Scrieți procedura, conform indicațiilor de mai sus.

Asemănător cu adunarea unui șir finit de numere să încercăm să realizăm o procedură care să efectueze produsul. De data aceasta variabila care va conține produsul va trebui inițializată cu 1 (deoarece dacă se inițializează cu 0 orice produs va fi tot 0).

Deci :

— inițializăm cu 1 variabila P (produsul) ;
— inițializăm cu 0 variabila NC (număr curent) ;
— repetăm de N ori :
  — adunarea lui „1" la NC (aflarea numărului actual) ;
  — înmulțirea lui NC cu P ;
— tipărim rezultatul final (P).

Scrieți procedura conform indicațiilor de mai sus. Dacă procedura nu „merge" puteți să vă ghidați după exemplul următor :

```
TO PROD :N
MAKE "P 1
MAKE "NC 0
REPEAT N [MAKE "NC :NC + 1 MAKE "P :NC * :P]
PRINT :P
END
```

UTILIZAREA COORDONATELOR

Coordonate

— Scrieți și testați procedura ACASĂ, care să aducă broasca acasă fără a trasa drumul parcurs.

— Scrieți și testați procedura MUTĂ, care să deplaseze broasca țestoasă cu X pixeli pe orizontală și Y pixeli pe verticală, fără a trasa drumul parcurs.

— Scrieți și testați procedura REPER, care să traseze o dreaptă orizontală și una verticală prin poziția inițială a broaștei (fig. 25), broasca avînd la sfîrșit aceeași poziție și orientare ca la început. Liniile vor fi trasate de la o margine la alta a ecranului care are dimensiunile din fig. 26.

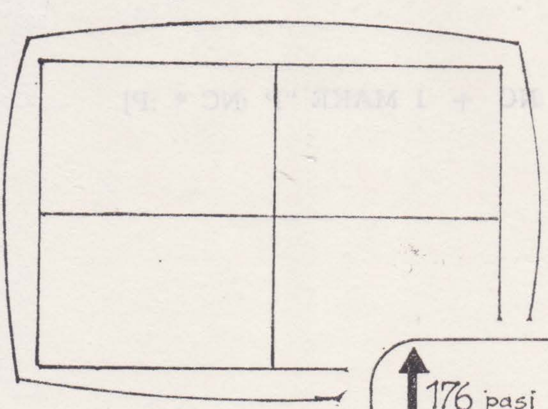


Fig. nr. 25

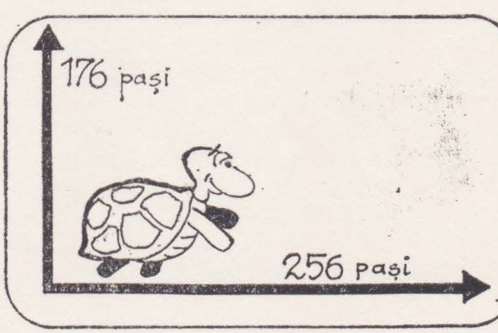


Fig. nr. 26

Procedura MUTĂ deplasează broasca țestoasă cu X pixeli pe orizontala broaștei și cu Y pixeli pe verticala acesteia. Dacă broasca pornește de „acasă", adică din poziția și orientarea inițială, orizontala ei coincide cu orizontala ecranului, iar verticala ei coincide cu verticala ecranului.

Ca să ducem broasca din centru într-un punct oarecare al ecranului folosind procedura MUTĂ, avem nevoie de cele două numere X și Y, care arată cu cît trebuie mutată broasca din centru „pe orizontală" și respectiv „pe verticală".

Cu alte cuvinte, putem spune că poziția unui punct de pe ecran poate fi **caracterizată** (sau „fixată", sau „definită") printr-un cuplu de două numere. Aceste două numere se numesc **coordonate**; primul se mai numește **abscisă** și se notează, de regulă, cu „x", iar al doilea se mai numește **ordonată**, și se notează de obicei cu „y".

Atunci cînd indicăm coordonatele unui punct, vom da întîi abscisa și apoi ordonata. De exemplu, dacă spunem că un punct A are coordonatele 20 și 50, se subînțelege că abscisa este 20 iar ordonata 50; în scris, pentru indicarea poziției unui punct se folosește notația de tipul A (20, 50).

Desenați pe ecran reperul creat de procedura cu același nume.

Mutați broasca în punctele următoare, aducînd-o acasă de fiecare dată cu procedura ACASĂ:

A (20,0) ; B (125,0) ; A (20,0) ; B (—125,0) ;
A (0,20) ; B (0,80) ; A (0, —20) ; B (0, —80) ;
A (80,50) ; B (—80,50) ; C (—80, —50) ; D (80, —50).

Reperul desenat de noi pe ecran este format din două drepte perpendiculare (care formează unghiuri drepte); pe ecran apare doar o parte a acestora, dar ne putem închipui cu ușurință că aceste drepte sînt nelimitate, fiecare în ambele sensuri. Un astfel de reper se mai numește **reper ortogonal** (de la „orto", cuvîntul grecesc care înseamnă „drept") sau reper cartezian (de la „Cartesius", forma latinizată a numelui lui **Descartes** (citim „decart"), matematicianul francez care a inventat acest reper).

Cele două drepte ale reperului nu sînt totuși niște „simple drepte"; pe fiecare din ele am stabilit prin convenție un sens pozitiv (la dreapta, respectiv în sus) și un sens negativ (la stînga, respectiv în jos). Am stabilit și o unitate de măsură a lungimilor (a distanțelor și a deplasărilor) care în cazul nostru este „pixelul". O dreaptă pe care s-a stabilit o origine, un sens pozitiv de parcurgere și o unitate de măsură se mai numește **axă.** Reperul cartezian este format deci din două axe ortogonale.

În plus, noi am stabilit și o „ierarhie" (sau o „ordine") a celor două axe: cea orizontală este „prima", iar cea verticală este „a doua".

Prin urmare:

Se numește **reper cartezian** o pereche ordonată de axe ortogonale, a căror origină comună este chiar punctul lor de intersecție.

Am văzut că față de un astfel de reper, poziția unui punct oarecare este fixată cu ajutorul a două numere numite „coordonate"; cele două coordonate ne arată **drumul** care trebuie parcurs din origine pînă în punctul dorit: prima parte a drumului (x-ul) se parcurge pe axa absciselor, iar a doua parte (y-ul) se parcurge ortogonal față de ea. De aceea, și coordonatele astfel definite se numesc coordonate „ortogonale" sau „carteziene".

Ce ordonată are un punct de pe axa absciselor?
Ce abscisă are un punct de pe axa ordonatelor?
Ce coordonate are centrul ecranului?

Funcții de poziție

Știm că ecranul este format din linii orizontale, fiecare linie fiind formată din pixeli; dimensiunea orizontală a ecranului este de 256 de pixeli, iar dimensiunea verticală a acestuia este de 176 pixeli.

Stabilindu-se ca originea reperului să fie în centrul ecranului, iar axele paralele cu laturile acestuia, rezultă că pixelii din marginea dreaptă au abscisa +127, iar cei din marginea stîngă au abscisa de —128; bineînțeles pixelii de pe axa verticală au abscisa egală cu 0. În mod analog, pixelii de pe latura superioară a ecranului au ordonata +87, iar cei de pe latura inferioară au ordonata —88; pixelii de pe axa orizontală au evident ordonata egală cu 0.

Limbajul LOGO admite această „înzestrare" a spațiului grafic cu reperul de care am vorbit mai sus; mai mult decît atît, limbajul conține numeroase primitive care admit ca intrări coordonate ale punctelor sau furnizează informații despre acestea.

Înainte de a trece în revistă cîteva din primitivele amintite, menționăm că există o comandă care permite broaștei să „iasă" din ecran mergînd în orice direcție maximum 32 767 de pași (pixeli). Cîmpul de mișcare al broaștei devine astfel deosebit de larg; ecranul apare ca o „fereastră" prin care se vede doar o mică parte din acest cîmp.

Comanda cu acest efect este:

WINDOW care înseamnă textual „fereastră"

După această comandă dacă broasca depășește limitele ecranului, ea nu mai reapare în partea opusă ci dispare pur și simplu de pe ecran. Dar această dispariție nu înseamnă că noi am pierdut „controlul" broaștei; ea se va afla exact unde i-am comandat și o putem deplasa în continuare, dacă nu ne deranjează că nu o vedem efectiv.

Mai mult decît atît putem să cunoaștem exact coordonatele broaștei; cele trei primitive care permit acest lucru sînt trei funcții (operații) LOGO:

XCOR — furnizează abscisa actuală a broaștei;

YCOR — furnizează ordonata actuală a broaștei;

POS — furnizează „poziția" broaștei, sub forma unei liste de două numere.

Aceste primitive fiind funcții LOGO trebuie să fie folosite numai ca intrări pentru alte comenzi sau operații. Pentru început, putem să cerem tipărirea valorilor respective, cu comanda PR (PRINT); de exemplu:
PR XCOR, PR YCOR, PR POS.

Dați comanda WINDOW! Rotiți broasca spre dreapta cu 30° și deplasați-o cu 50 de pași înainte. Cereți afișarea abscisei broaștei; cereți tipărirea ordonatei. Cereți afișarea poziției actuale a broaștei.

Deplasați broasca înainte cu încă 200 de pixeli; afișați poziția ei.

Știm că „broasca țestoasă" nu este un simplu „punct"; ea se poate **roti** într-o parte sau alta, avînd deci diferite „orientări".

Se numește **unghi de orientare** al broaștei (sau mai scurt, **orientarea** ei) unghiul format de direcția broaștei cu paralela pusă prin ea la axa ordonatelor (axa verticală). Unghiul de orientare se măsoară de la „nord" spre „est", luînd valori între 0° și 359°.

Funcția LOGO HEADING (adică „orientare") ne furnizează valoarea actuală a orientării broaștei.

Aduceți broasca acasă cu procedura ACASĂ. Deplasați-o pe un drum „frînt", format din trei segmente; după fiecare deplasare afișați orientarea actuală a broaștei.

Dacă vrem să aflăm ce orientare trebuie să aibe broasca pentru a fi îndreptată spre un anumit punct apelăm funcția:
TOWARDS [x y] (înseamnă „către")

Aici evident x și y sînt coordonatele punctului vizat.

Dați comanda PR TOWARDS [0 0], pentru a afla ce orientare ar trebui să aibe broasca pentru a fi îndreptată spre casă (origine). Observați că afișarea unghiului de orientare nu modifică în nici un fel orientarea însăși a broaștei.

Dintre comenzile care produc modificări în orientarea sau poziția broaștei amintim pe:
SETH n (SETHEADING n) — „pune orientarea!"

Această comandă nu schimbă locul broaștei pe ecran, dar o rotește astfel încît să capete orientarea egală cu n. Dar în locul intrării „n" putem pune și o funcție care să ne furnizeze un unghi de orientare; de exemplu comanda: SETH TOWARDS [50 70] va orienta broasca astfel încît să fie îndreptată spre punctul de coordonate 50 și 70.

Orientați broasca spre origine!

Repetînd comanda FD n de mai multe ori, încercați să aduceți broasca în origine; încercați să „măsurați" astfel drumul de revenire acasă.

ALTE OBIECTE LOGO

Liste, cuvinte, caractere

În paragraful precedent am întîlnit o listă, cea care caracterizează poziția unui punct ; evident este vorba de lista coordonatelor punctului respectiv. Limbajul LOGO poate să opereze și cu liste care au mai mult de două elemente ; ceea ce este foarte interesant și demn de reținut din primul moment — faptul că **fiecare listă LOGO** este memorată într-o **variabilă.** Iată deci că variabilele pot avea conținuturi mult mai diverse decît ne-am închipuit pînă acum.

Ne putem convinge imediat de justețea afirmației de mai sus dînd comenzile CS MAKE "P POS PR :P ; pe ecran va fi afișat conținutul variabilei P, adică lista coordonatelor broaștei aflată „la domiciliu" 0 0. Dacă mutăm broasca în altă parte, aceleași comenzi ne vor permite să afișăm noile coordonate ale broaștei ; să reținem faptul important de care ne-am convins : orice listă poate fi memorată într-o variabilă.

Dar listele LOGO pot să conțină nu numai **numere,** ci și **cuvinte ;** este necesar ca în legătură cu toate acestea să facem cîteva precizări :

— **Obiectele LOGO** sînt cuvinte sau liste utilizate ca intrări sau ieșiri ale procedurilor ;

— Un **cuvînt** este o serie de caractere alfabetice sau numerice ; pentru a fi deosebite de numele de proceduri, înaintea cuvintelor se pun **ghilimele ;** un cuvînt nu poate cuprinde **caracterul „spațiu",** deoarece aceasta indică terminarea cuvîntului ; există și **cuvîntul vid,** scris doar cu ghilimele, fără nimic după ele ;

— **Numerele** sînt de asemenea cuvinte dar ele se pot scrie fără " ;

— Cuvintele care exprimă valori logice (TRUE — adevărat și FALSE — fals) pot fi scrise, și ele fără ghilimele ;

— O **listă** LOGO constă dintr-o serie de obiecte LOGO, adică de cuvinte sau alte liste ; o listă este cuprinsă între paranteze drepte ([,]) ; elementele unei liste sînt separate prin spații ; **lista vidă** (fără nici un element) este [ ] ;

— După cum știm, cuvintele pot fi utilizate și ca **nume de variabile ;**

— **Elementele** unui cuvînt sînt caractere, iar elementele unei liste sînt cuvinte sau liste ;

— Atît cuvintele cît și listele pot fi „depuse" în variabile, fie prin instrucțiunea de atribuire MAKE fie prin alte procedee.

Depuneți în variabilele A, B, C și D următoarele conținuturi și dați de fiecare dată comanda de tipărire a conținutului variabilei respective : 32.26 ; [3 5 7]; "MERE ; [PE MINE MĂ CHEAMĂ . . . .].

Operații cu liste, cuvinte și caractere

Reamintim că primitivele de mai jos fiind **funcții** (operații) LOGO, trebuie să fie utilizate ca intrări pentru alte comenzi sau operații (PR, MAKE etc.).

COUNT obiect (numără !) — furnizează numărul de elemente al obiectului dat ;

ITEM n listă (articol, element) — furnizează al n-lea element al listei ;

LAST obiect (ultimul) — dă ultimul element ;

FIRST obiect (primul) — dă primul element ;

BF obiect (BUTFIRST, fără primul) furnizează obiectul fără primul element ;

BL obiect (BUTLAST, fără ultimul) — furnizează obiectul fără ultimul element.

Afișați numărul de elemente al fiecărui obiect creat în exercițiul precedent.

Afișați primul element al acestor obiecte.

Depuneți într-o variabilă conținutul variabilei D, mai puțin ultimul element.

Afișați al treilea element al noului obiect.

Uneori apare necesitatea ca din mai multe caractere sau cuvinte să formăm un nou cuvînt sau ca din mai multe cuvinte să formăm o listă. Următoarele primitive (tot funcții !) rezolvă această cerință :

WORD obiect1 obiect2 sau WORD (obiect1 obiect2 . . . obiectn) (cuvînt) — furnizează un cuvînt format prin alipirea obiectelor date ; acestea trebuie să fie numai caractere sau cuvinte ;

SE obiect1 obiect2 sau SE (obiect1 obiect2. . . . obiectn) (SENTENCE, propoziție) — furnizează o listă avînd ca element obiectele date.

Primitiva SE are o mare importanță deoarece multe instrucțiuni LOGO cer o listă ca parametru de intrare ; ori deseori elementele din care dorim să formăm lista respectivă sînt stocate în diferite variabile sau urmează să rezulte dintr-un calcul. Astfel, de exemplu deși sînt delimitate de paranteze drepte, [:A :B] și [5 7+RANDOM 20] nu

sînt liste, deoarece elementele lor nu sînt cuvinte sau liste; putem forma listele corecte astfel: SE :A :B; SE 5 7+RANDOM 20.

Un exemplu mai elocvent: vrem să orientăm broasca țestoasă cu capul spre un punct ale cărui coordonate sînt depuse în variabilele :X și :Y; dacă scriem SETH TOWARDS :X :Y, calculatorul nu execută comanda și dă mesajul de eroare „TOWARDS nu merge cu :X ca input".

Soluția corectă este: SETH TOWARDS SE :X :Y

Următoarele două primitive permit să adăugăm un nou element la o listă constituită anterior:

LPUT obiect listă — furnizează o nouă listă formată din cea dată, punîndu-i ca **ultim** element obiectul dat;

FPUT obiect listă — furnizează o nouă listă, formată din cea dată, punîndu-i ca **prim** element obiectul dat.

În sfîrșit, următoarele două primitive permit introducerea de la tastatură a unor obiecte LOGO, în timpul execuției unei proceduri:

RC (READCHAR) — (citește caracter!) — comandă calculatorul să aștepte apăsarea unei taste și apoi furnizează caracterul introdus; acesta trebuie să fie „luat în primire" de procedură, ca orice rezultat al unei operații. De exemplu:

MAKE "A RC   introduce caracterul citit în A.

RL (READLIST) — (citește lista!) — furnizează lista dată de utilizator la tastatură;

MAKE "L RL   depune în variabila L lista citită.

Citiți o listă de la tastatură. Citiți trei caractere de la tastatură. Formați din aceste caractere un cuvînt. Adăugați acest cuvînt la începutul listei. Adăugați-l și la sfîrșit.

Explicați acțiunea procedurilor de mai jos, introduceți-le în memorie și verificați funcționarea lor:

```
TO REPDES
MAKE "L [ ]
 DESEN CS WAIT 100 COPIE :L
END
```

Comanda WAIT 100 cere calculatorului să aștepte timp de două secunde.

```
TO DESEN
MAKE "C RC
MAKE "L LPUT :C :L
IF :C="V [FD 5]
IF :C="B [RT 5 FD 2]
IF :C="N [RT 15 FD 2]
IF :C="M [RT 25 FD 2]
IF :C="C [LT 5 FD 2]
IF :C="X [LT 15 FD 2]
IF :C="Y [LT 25 FD 2]
IF :C="S [STOP]
DESEN
END
```

```
TO COPIE :L
MAKE "C FIRST :L
MAKE "L BF :L
(toate IF-urile din DESEN)
COPIE :L
END
```

Bineînțeles, vom realiza procedura COPIE schimbînd numele procedurii DESEN și operînd modificările necesare în interiorul ei.

Remarcăm că procedura COPIE „distruge" doar lista din variabila **locală** L, în timp ce lista din variabila **globală** L, creată cu REPDES și DESEN, rămîne intactă. În consecință, după alte „ștergeri" ale ecranului, desenul poate fi refăcut cu comanda COPIE :L ori de cîte ori dorim.

Introduceți în DESEN și COPIE și alte **posibilități de manevrare** a broaștei !

Manevrarea „manuală" a broaștei țestoase

În multe proiecte se dovedește utilă existența unor proceduri care să ne permită manevrarea „manuală" a broaștei țestoase, adică mișcarea acesteia prin simpla apăsare a unor taste. O primă observație se impune : astfel de proceduri **trebuie să se autoapeleze,** pentru ca acțiunea lor să dureze cît dorește utilizatorul. În consecință, ele trebuie să prevadă și posibilitatea **opririi** autoapelării, prin apăsarea unei taste pe care ne-o alegem de la început, în mod convențional ; să alegem, pentru aceasta, tasta „S" (de la „STOP"). Structura unei astfel de proceduri va fi, deci :

```
TO (nume procedură)
MAKE "T RC
(comenzile de mișcare)
IF :T = "S [STOP] [nume procedură]
END
```

Am notat cu T variabila în care se depune caracterul furnizat de funcția RC, care „preia" tasta apăsată de noi.

Scrieți și testați o procedură, numită ROTM, care să rotească broasca la dreapta sau la stînga cu cîte 5°, ori de cîte ori se apasă tasta R sau L (de la „right", respectiv „left").

Ne propunem acum să scriem o astfel de procedură, care să conducă broasca înainte, pe direcția ei, la apăsarea unei taste numerice, cu atîția pași, cît indică tasta numerică apăsată.

Deci, **dacă** tasta apăsată este numerică, broasca va înainta cu pașii respectivi. În LOGO există o funcție cu valori logice, care ia valoarea „adevărată" sau „fals", după cum argumentul ei este numeric sau nu. Această funcție este NUMBERP. Folosind funcția aceasta procedura noastră va arăta astfel :

```
TO INM
MAKE "T RC
IF NUMBERP :T [FD :T]
IF :T = "S [STOP] [INM]
END
```

În utilizarea procedurii este posibil ca urmărind înaintarea broaștei, să nu ne oprim la timp; ar fi de dorit (în acest caz) să ne putem corecta, retrăgînd broasca. Pentru aceasta, vom introduce, după **primul IF**, instrucțiunea :

```
IF :T = "B [PE BK 5 PD]
```

care prevede ca, la apăsarea tastei „B", broasca să se retragă cu 5 pași, ștergînd linia trasată.

În unele cazuri, este foarte util ca, deplasînd broasca, să și „măsurăm" drumul parcurs ; pentru aceasta, va trebui ca, într-o variabilă globală (fie ea LD — lungimea drumului), să adunăm (sau să scădem) fiecare deplasare.

Evident, LD se va inițializa cu 0 înainte de fiecare apelare a procedurii INM. În procedură, **după fiecare** comandă de deplasare (FD sau BK), vom introduce modificarea valorii lui LD, corespunzătoare deplasării respective. În final, procedura noastră va arăta astfel:

```
TO INM
MAKE "T RC
IF NUMBERP :T [FD :T MAKE "LD :LD + :T]
IF :T = "B [PE BK 5 PD MAKE "L :LD — 5]
IF :T = "S [STOP] [INM]
END
```

Apelarea procedurii se va face astfel :
MAKE "LD 0 INM   urmat, eventual, de :

PR :LD SAU MAKE "variabilă :LD, dacă dorim să „memorăm" lungimea drumului.

Testați procedura de mai sus, după ce ați rotit în prealabil broasca, manual sau prin comenzi LOGO.

PROBLEME ȘCOLARE

Construcții de triunghiuri

Triunghiurile desenate de noi pînă acum aveau o caracteristică cu totul particulară : erau echilaterale, adică aveau toate laturile congruente ; de asemenea, „ni s-a spus" că ele au toate unghiurile de 60°. Ne propunem să desenăm acum alte tipuri de triunghiuri, mai puțin particulare.

Vom remarca de la început că pentru a desena un triunghi, nu este necesar să cunoaștem **toate** elementele acestuia (toate laturile și toate unghiurile).

Să considerăm, astfel, cazul în care cunoaștem doar un unghi și cele două laturi ale acestuia, cum se spune, mai corect, două laturi și unghiul cuprins între ele. Să notăm cele trei intrări cu L1, U și L2. Procedura se va numi LUL (de la „latură — unghi—latură").

Ideea construcției este simplă :

— trasăm latura L1 (de la „coadă" spre vîrful unghiului U);

— rotim broasca țestoasă astfel ca noua ei direcție să facă un unghi U cu cea precedentă ; evident, rotația va fi de 180° — :U ;

— trasăm a doua latură (L2) ;

— orientăm broasca spre punctul inițial.

— unim punctul final cu cel inițial.

Este evident că, pentru a realiza aceste acțiuni, trebuie să „memorăm" punctul inițial cu :

ST MAKE "P1 POS ;

orientarea broaștei spre punctul inițial se face cu :

SETH TOWARDS :P1.

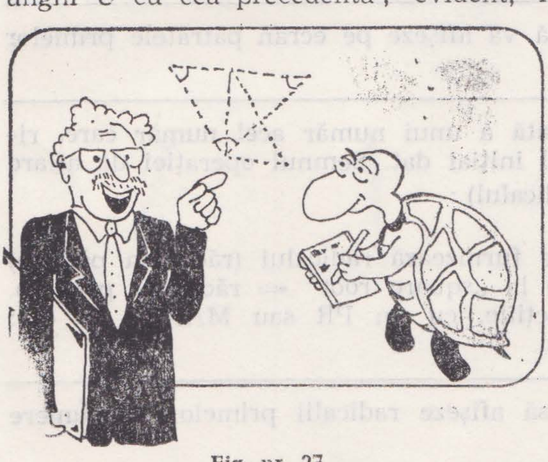


Fig. nr. 27

Unirea punctului final cu cel inițial s-ar fi putut face, simplu, cu SETPOS :P1, care ne-ar fi „închis" triunghiul ; totuși astfel nu am fi putut afla lungimea celei de a treia laturi. De aceea preferăm să închidem „manual" triunghiul, cu :

MAKE "LD 0 INM MAKE "L3 :LD PR :L3

care ne și furnizează cea de a treia latură.

Scrieți și testați procedura LUL, avînd ca intrări pe L1, U și L2, pe baza indicațiilor de mai sus.

Arii, pătrate și radicali

Aria unei suprafețe este un **număr** care ne arată de cîte ori „intră" în suprafața respectivă un pătrat cu latura de o unitate. Cel mai simplu este să calculăm aria unui dreptunghi ; dacă dreptunghiul are baza B și înălțimea I, el va putea fi acoperit de B coloane de cîte I pătrate cu latura 1. De exemplu, dacă B = 7 cm și I = 3 cm dreptunghiul poate fi „acoperit" cu 7 × 3 = 21 de pătrate cu latura 1 cm. În general, deci, aria unui dreptunghi este dată de formula :

$$A = B \cdot I$$

În cazul unui pătrat cu latura L, judecăm în felul următor : pătratul respectiv este un dreptunghi, pentru care B = L și I = L, deci :

$$A = L \cdot L$$

Produsul unui număr cu el însuși se mai numește „ridicare la pătrat" și se notează cu $L \cdot L = L^2$. Originea acestei denumiri este tocmai operația de aflare a ariei... unui pătrat, din care rezultă operația aritmetică respectivă.

Scrieți o procedură care să vă afișeze pe ecran pătratele primelor N numere naturale.

Se numește **rădăcină pătrată** a unui număr acel număr care, ridicat la pătrat, ne dă numărul inițial dat. Semnul operației de aflare a rădăcinii pătrate este $\sqrt{\ }$ (radicalul) :

$\sqrt{25} = 5$, pentru că $5^2 = 25$.

În LOGO, funcția care ne furnizează radicalul (rădăcina pătrată) a unui număr este SQRT (de la „square root" = rădăcină pătrată). Ea se folosește, ca toate funcțiile, cu un PR sau MAKE ; de ex : PR SQRT 25, PR SQRT :A etc.

Scrieți o procedură care să afișeze radicalii primelor N numere naturale.

Să facem, acum, cîteva observații privitoare la semne ; știm că „semnul" unui număr se schimbă în contrariul său dacă înmulțim acel număr cu „—1" : 5 . (—1)=—5, (—5) . (—1)=5 etc... de aici re zultă un fapt foarte important : **pătratul** oricărui umăr (pozitiv sau negativ) este **întotdeauna pozitiv.**

Într-adevăr, de exemplu : $(-7)^2 = (-7)\cdot(-7) = (-1)\cdot 7\cdot(-1)\cdot 7 = 7 . 7=49$. Pentru numerele pozitive nu mai avem nevoie de nici o demonstrație.

De aici rezultă că **numai numerele pozitive au rădăcini pătrate** (radicali) ; într-adevăr, dacă dați comanda :

PR SQRT — 25, veți primi un mesaj de eroare.

Pe de altă parte, orice număr pozitiv va avea **două** rădăcini pătrate ; de exemplu :

(—5) . (—5)=25, dar și 5 . 5=25 ; 3 . 3=9, dar și (—3) . (—3)=9 etc.

Prin convenție, rădăcina pătrată **pozitivă** se numește rădăcina pătrată **aritmetică** sau **radicalul aritmetic** al unui număr. Acum putem face precizarea că **funcția SQRT furnizează radicalul artimetic al argumentului.**

Astfel, dacă ne e „lene" să scriem o procedură pentru aflarea valorii absolute a unui număr, cu luarea în considerare a celor trei cazuri (număr negativ, nul sau pozitiv), putem afla valoarea absolută astfel : RP SQRT :A * :A.

Verificați, pentru mai multe numere, pozitive și negative, acest mod de aflare a valorii absolute (a „modulului").

Determinarea grafică a unghiurilor

Știm că funcția TOWARDS :P ne furnizează **unghiul de orientare** („heading"-ul) pe care ar trebui să-l aibe broasca spre a fi îndreptat către punctul ale cărui coordonate sînt „stocate" în variabila P. Mai știm că, în LOGO, unghiul de orientare se măsoară **de la Nord spre Est.**

Să considerăm, acum, că broasca se află într-un punct P ; mai considerăm două puncte, P1 și P2, distincte și diferite de P. Ne propunem să aflăm măsura unghiului (P1) P (P2), adică a unghiului U format de dreptele PP1 și PP2.

Făcînd legătura cu funcția TOWARDS, ne dăm seama că unghiul căutat, P, este tocmai diferența unghiurilor de orientare către P2 și către P1 ; deci, problema se rezolvă prin :

MAKE "U (TOWARDS :P2) — TOWARDS :P1.

Dar se poate întîmpla ca punctele P1 și P2 să fie așezate „invers", și atunci obținem aceeași valoare, dar cu semnul minus ; după instrucțiunea de mai sus, va trebui să punem o comandă de înlocuire a lui U cu valoarea sa absolută ; de exemplu :

MAKE "U SQRT :U * :U

În sfîrșit, se poate întîmpla ca punctele să fie foarte „prost" plasate ; de exemplu, PP1 să fie spre E de direcția PN (nord), iar PP2 să fie spre W de această direcție. În acest caz, unghiul obținut prin calculul de mai sus va fi . . mai mare de 180° ; noi considerăm, însă, că unghiul celor două direcții este celălalt, care nu depășește 180°. Cele două instrucțiuni de mai sus trebuie să fie, deci, urmate de :

IF :U >180 [MAKE "U 360 — :U]

Reuniți cele de mai sus într-o procedură numită UNG, care are intrările :P1 și :P2.

Completați procedura LUL astfel încît ea să poată „calcula" și cele trei unghiuri ale triunghiului desenat.

ATENȚIE : va trebui să „memorați" toate vîrfurile triunghiului. După desenare veți plimba broasca (cu SETPOS) în cele trei vîrfuri și veți apela în mod corespunzător procedura UNG. Tipăriți cele trei laturi și cele trei unghiuri !

Relația lui Pitagora

Construind triunghiul pe baza a trei elemente (două laturi și un unghi) am putut constata că între cele șase elemente ale triunghiului există unele relații (legături). Într-adevăr, fiind date cele trei elemente cu care am construit triunghiul, celelalte trei elemente **au rezultat automat** : ele nu puteau fi **altele** decît cele obținute de noi ! Deși nu ne dăm încă seama ce formă concretă au aceste relații dintre elementele unui triunghi, ne dăm totuși seama că ele **există**.

Vom încerca acum să punem în evidență o astfel de relație, care leagă între ele laturile unui triunghi dreptunghic. Mai întîi, să facem cîteva precizări privind termenii utilizați : într-un triunghi dreptunghic, latura care se opune unghiului drept se numește **ipotenuză,** iar laturile alăturate unghiului drept se numesc **catete.**

Să considerăm un triunghi dreptunghic cu catete cunoscute. **În** acest caz triunghiul va putea fi construit cu procedura LUL, luînd pentru unghiul U valoarea de 90°. După construirea unui triunghi dreptunghic, vom cunoaște și cea de a treia latură (ipotenuza), care este determinată de subprocedura INM ; numele ipotenuzei este :L3.

Vom completa procedura astfel încît, înainte de terminare, ea să tipărească, sub forma unei liste, două valori : **pătratul ipotenuzei** și **suma pătratelor catetelor ;** punem, deci, înainte de END, linia LOGO :

PR SE :L3 * :L3 :L1 * :L1 + :L2 * :L2

Rulați de mai multe ori procedura. Aveți grijă să dați mereu valoarea de 90° pentru unghi !

Ce constatați privitor la rezultatele afișate ?

Dacă ați „închis" cu atenție triunghiul, puteți constata că **pătratul ipotenuzei este egal cu suma pătratelor catetelor** ; micile diferențe dintre cele două mărimi tipărite se datoresc lipsei de „finețe" în închiderea triunghiurilor, care provoacă erori amplificate prin ridicarea la pătrat.

Oricum, dacă luaţi cîteva exemple cu date foarte diferite, vă puteţi convinge de adevărul celor spuse. Dar acest adevăr poate fi „demonstrat", adică dedus din alte adevăruri matematice, mai simple. De aceea, propoziţia de mai sus se numeşte **„teoremă"** ; ea poartă numele primului matematician care a demonstrat-o : Pitagora. S-o mai enunţăm o dată :

În orice triunghi dreptunghic, pătratul ipotenuzei este egal cu suma pătratelor catetelor.

Relaţia afirmată de teorema lui Pitagora ne permite să calculăm ipotenuza dacă cunoaştem catetele : calculul constă în **extragerea radicalului** din suma pătratelor catetelor. Bineînţeles, dacă se cunoaşte ipotenuza şi o catetă, putem afla cealaltă catetă.

Distanţa dintre două puncte

Să presupunem că ni s-au dat două puncte, prin coordonatele lor, cuprinse în două liste ; în fiecare listă, primul element este abscisa (x-ul), iar al doilea (ultimul) element este ordonata (y-ul). Ne propunem să scriem o procedură care, primind aceste informaţii, să ne furnizeze **distanţa** dintre cele două puncte.

Pentru aceasta vom face, mai întîi, un desen care să ne permită înţelegerea şi rezolvarea problemei ; ne propunem să desenăm cele două axe, să „plasăm" cele două puncte în poziţiile date ca intrări, să le unim şi, apoi, să trasăm două linii paralele cu Ox, respectiv Oy, astfel încît să se formeze un triunghi dreptunghic :

```
TO DESEN :P1 :P2                 SETPOS :P2
RT 90 FD 256 LT 90 FD 176        SETY LAST :P1
DOT :P1 DOT :P2                  SETPOS :P1
PU SETPOS :P1 PD                 END
```

Rulaţi procedura de mai sus, dînd ca intrări [10 10] [70 30] şi altele, care să menţină triunghiul întreg pe ecran.

Analizînd triunghiul din figura rezultată observăm că distanţa dintre punctele date este chiar ipotenuza triunghiului ; cateta orizontală este tocmai diferenţa absciselor, iar cateta verticală este diferenţa ordonatelor. Deoarece coordonatele (abscisele şi ordonatele) sînt cunoscute, catetele se află imediat, iar teorema lui Pitagora ne permite să aflăm ipotenuza, adică distanţa dintre cele două puncte date.

Vom scrie procedura ca pe o funcţie, adică rezultatul va fi dat prin OP (OUTPUT), nu prin memorare într-o variabilă globală :

```
TO DIST :P1 :P2
MAKE "DX (FIRST :P2) — FIRST :P1
MAKE "DY (LAST :P2) — LAST :P1
OP SQRT (:DX * :DX + :DY * :DY)
END
```

Testaţi funcţia DIST cu mai multe cupluri de puncte date (comandaţi PR DIST . . . . !).

Poligoane regulate

Ne amintim că am realizat cîndva o procedură numită STEA, care desena o ,stea" formată din N raze de lungime L fiecare; înainte de a trasa steaua, calculăm (în procedură!) mărimea unghiului cu care se va roti broasca de la o rază la alta. Ţinem seama de faptul că broasca se va roti, în total, cu 360° şi că toate unghiurile (N la număr) sînt egale.

Scrieţi procedura STEA şi testaţi-o cu diferite valori pentru N şi L.

Să schimbăm numele procedurii în POLI, lăsîndu-i aceleaşi intrări, dar ştergind din ea comanda de retragere a broaştei, obţinem :

```
TO POLI :N :L
MAKE "U 360/ :N
REPEAT :N [FD :L RT :U]
END
```

Testind procedura constatăm că ea nu mai desenează o stea (era normal!), dar sîntem surprinşi de faptul că desenează un contur **închis**; mai precis, este vorba de un **poligon regulat** cu N laturi.

Poligonul este o „linie frîntă închisă", iar acesta se numeşte **regulat**, deoarece are toate laturile congruente şi, în plus, toate unghiurile congruente. Şi în legătură cu unghiurile poligonului mai avem ceva de spus : după fiecare înaintare, broasca s-a rotit cu unghiul format de prelungirea unei laturi cu latura următoare. Un astfel de unghi se mai numeşte **unghi exterior al poligonului.** Dar noi ştim că broasca s-a rotit, în total, cu 360° ; deci :

Suma unghiurilor exterioare ale poligonului este de 360°, dacă poligonul este parcurs într-un singur sens.

Noi am notat cu U mărimea unghiului exterior ; unghiul „interior" are, deci, mărimea de 180° — U. Prin urmare, suma unghiurilor interioare este de N . (180—U) = N . 180 — N . U ; dar N . U = 360, deci suma unghiurilor este de N . 180 — 360 = 180 . (N—2) !

Testaţi procedura POLI dînd diferite valori lui N şi L!

Să facem o observaţie în legătură cu **poziţia** poligonului desenat de procedura POLI ; este uşor de văzut că poligonul are o parte „mai mare" în partea de sus a ecranului decît în partea de jos. Pentru a ne convinge, să rotim broasca la orizontală şi să desenăm orizontala ecranului :

```
RT 90 FD 256
```

Este evidentă asimetria aşezării poligonului faţă de orizontala broaştei ; pentru ca poligonul să "iasă" simetric faţă de orizontala broaştei, vom introduce în procedură, înainte de REPEAT, rotirea broaştei cu :U/2 (RT :U/2), iar după linia cu REPEAT readucem broasca în poziţia iniţială (LT :U/2). Să numim noua procedură POLIS :

```
TO POLIS :N :L
MAKE "U 360/:N
RT :U/2
REPEAT : N [FD : L RT : U]
LT : U/2
END
```

Testați procedura POLIS pentru N=10 și L=40 ; trasați orizontala broaștei.

În sfîrșit, o ultimă problemă legată de poligoanele rezultate : prin datele pe care le furnizăm procedurii, noi determinăm „mărimea" poligonului, dar nu știm dinainte cît de mult se va „intinde" el pe ecran. Să încercăm întîi să exprimăm mai bine ideea de „mărime" a poligonului.

Ați observat, probabil, că vîrfurile poligonuui par a fi așezate pe un cerc ; de fapt, dacă vrem să desenăm un poligon regulat pe hîrtie, chiar așa procedăm : trasăm un cerc, îl împărțim (cu raportorul) în N părți egale și unim apoi, la rînd, punctele respective de pe cerc.

Revenind la problema „mărimii" poligonului regulat, ne dăm seama că ea poate fi caracterizată chiar prin raza cercului pe care se află vîrfurile poligonului.

Rezultă că, dacă vrem să desenăm un poligon regulat cu N laturi, care să „încapă" într-un cerc de rază dată, R, va trebui să determinăm în mod adecvat lungimea laturii L (să nu o mai luăm, deci, arbitrară).

Latura se poate determina foarte simplu : considerăm broasca în centrul cercului ; o deplasăm pe cerc (FD : R), memorăm punctul în care am ajuns, aducem broasca la loc, o rotim cu unghiul : U (360/: N) și o deplasăm iarăși pe cerc, tocmai în vîrful următor al poligonului. Memorăm acest al doilea vîrf și calculăm distanța dintre cele două puncte memorate, apelînd procedura — funcție DIST.

Să scriem, deci, procedura de desenare a poligonului regulat cu N laturi, inscriptibil într-un cerc de rază R (broasca pleacă din și revine în centrul poligonului) :

```
TO POLIR :N :R              MAKE "L DIST :P1 :P2
PU FD :R MAKE "P1 POS       LT 90 FD :R RT 90 PD
BK :R RT 360/:N             POLIS :N :L
FD :R MAKE "P2 POS          PU RT 90 FD :R LT 90 PD
BK :R LT 360/:N             END
```

Scrieți și testați procedura POLIR cu mai multe valori date intrărilor N și R.

Cercuri mari și mici

Ați observat, probabil, că dacă numărul laturilor este mare, poligonul obținut seamănă cu un ... cerc; dacă de exemplu luați N=60, poligonul desenat nu mai poate fi deosebit de un cerc, bineînțeles în limitele graficii de pe ecran ! Prin urmare, pentru desenarea pe ecran a unui cerc de rază R, vom da comanda :

POLIR 60 :R

Totuși, dacă cercul desenat are raza mică, începe să ne deranjeze faptul că durata desenării este prea mare în raport cu mărimea cercului și, de asemenea, observăm că, din cauza aproximațiilor de calcul, cerculețul desenat este prea „colțuros",

---

Desenați cercuri de rază 10, 5, 3!

Pentru a remedia aceste neajunsuri, trebuie să reducem volumul de calcule și, de asemenea, numărul de laturi, dacă 60 este deja prea mare. Pentru aceasta, vom schimba, mai întîi, modul de calcul al laturii poligonului care „materializează" cercul.

Pornim de la faptul, pe care probabil că îl cunoașteți, că lungimea cercului se află înmulțind raza cu 2 și cu 3,14. În acest caz, latura poligonului se va afla împărțind lungimea cercului la numărul de laturi.

Cît privește numărul de laturi, vom proceda în felul următor: dacă lungimea cercului este mai mare de 60 de pași, vom lua ca număr de laturi tot 60, ca mai înainte. Dacă lunigmea cercului este mai mică decît 60, vom lua ca număr de laturi .... partea întreagă a lungimii cercului. Prin aceasta, evităm situația, inutilă, în care latura poligonului ar fi mai mică decît un pas.

Deci, procedura astfel concepută va desena la fel de bine atît cercurile mari cît și cele mici. Să-i scriem textul, conform cu cele de mai sus:

```
TO CERC :R
MAKE "LC 2 * 3.14 * :R
IF :LC < 60 [MAKE "N INT :LC] [MAKE "N 60]
MAKE "L : LC/:N
MAKE "U 360/:N
PU FD :R RT 90 + :U/2 PD
REPEAT :N [FD :L RT :U]
LT 90 + :U/2 PU BK :R PD
END
```

Testați procedura CERC desenînd cercuri de diferite raze !

MODELE GRAFICE DECORATIVE

O altă stea

„Steaua" desenată de noi pînă acum a fost destul de puțin „arătoasă" ; pentru a obține o stea propriu-zisă, vă propunem mai întîi să scrieți o procedură pentru desenarea unui **romb** de latură L, avînd mărimea unghiului ascuțit :U :

```
TO ROMB :L :U
REPEAT 2 [FD :L RT :U FD :L RT 180 — :U]
END
```

Mai departe, ne închipuim steaua formată din romburi lipite care pornesc toate din același centru ; datele de intrare vor fi : numărul de „colțuri" (romburi) și latura fiecărui romb. Prin urmare, procedura STEA va calcula unghiul la centru (360/:N) și va desena :N romburi, fiecare fiind rotit față de cel precedent cu unghiul U, calculat ca mai sus (vezi fig. 28).

Scrieți și testați procedura STEA, conform indicațiilor de mai sus !

Un model cu repetiții și creșteri

Să ne reamintim TRAP, care desenează un anumit trapez de bază (mare) L :

```
TO TRAP :L
RT 30
REPEAT 3 [FD :L/2 RT 60]
RT 60 FD :L RT 90
END
```

Scrieți și testați procedura de mai sus !

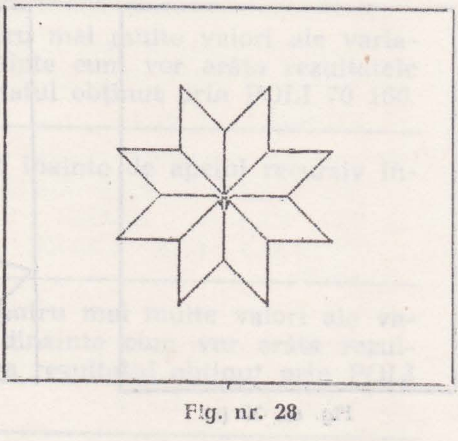


Fig. nr. 28

Ne reamintim, de asemenea, că pornind de la o figură oarecare (dreptunghi, triunghi, steag etc.), am creat diverse modele decorative, prin simpla desenare repetată a figurii, rotită în mod corespunzător în jurul punctului inițial.

Vă propunem acum să rotim trapezul de mai sus, dar, la fiecare nouă desenare a trapezului, să mărim latura cu 2 pași. Ca date de intrare vom avea : latura inițială a trapezului (L), unghiul de rotire (U) și numărul de semirotații complete (NS). Numărul de trapeze desenate va fi, evident, partea intreagă a expresiei 180 * :NS/:U.

Procedura va fi:

```
TO ROCR :L :U :NS
MAKE "N 180 * :NS/: U
REPEAT :N [TRAP :L RT :U MAKE "L :L + 2]
END
```

Scrieți și testați procedura de mai sus. Luați de exemplu, L = 1, U = 12, NS = 3 (fig. 29). Ce imagine obțineți ?

Cum ar fi trebuit, deci, să denumim procedura ?

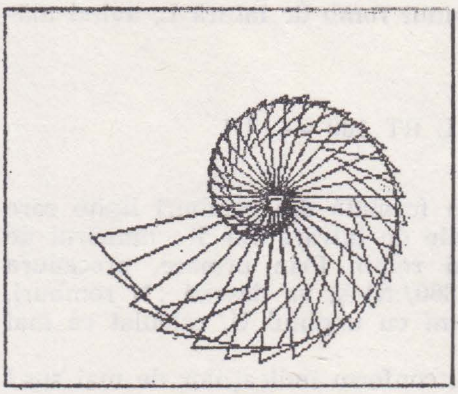


Fig. nr. 29

Realizarea de modele folosind apelarea recursivă

Apelarea recursivă poate avea în LOGO rezultate grafice deosebit de spectaculoase. În același timp, însă, apelurile recursive sînt mai „mîncătoare de memorie", adică consumă o parte destul de mare din memoria pe care LOGO o pune la dispoziție pentru memorarea proce-

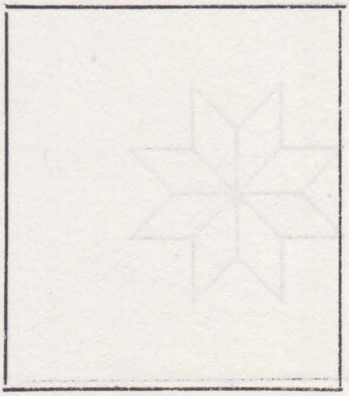


Fig. nr. 29 (a)

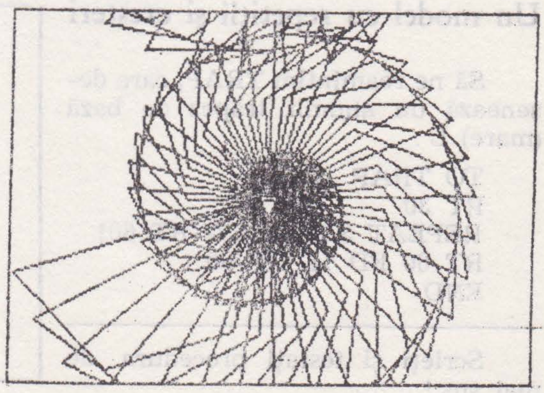


Fig. 29 (b)

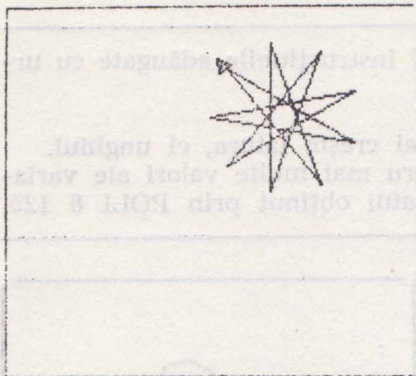


Fig. nr. 30

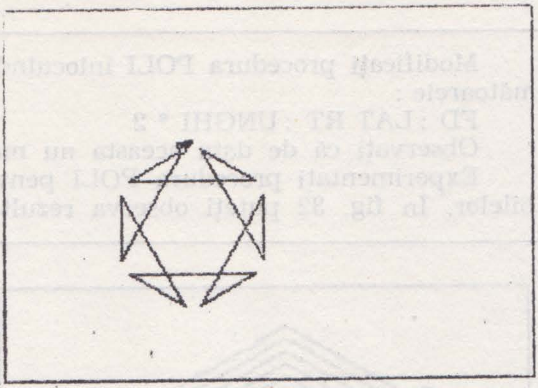


Fig. nr. 31

durilor. Pentru ca într-o procedură apelarea recursivă să fie cît mai eficientă (adică să „mănînce" cît mai puțină memorie), **apelarea recursivă trebuie pusă „la coadă"**, adică înainte de cuvîntul END care termină orice procedură.

Procedura POLI are două variabile care determină mărimea (:LAT) și respectiv forma (:UNGHI) figurii și, deasemenea, folosește apelul recursiv la coadă :

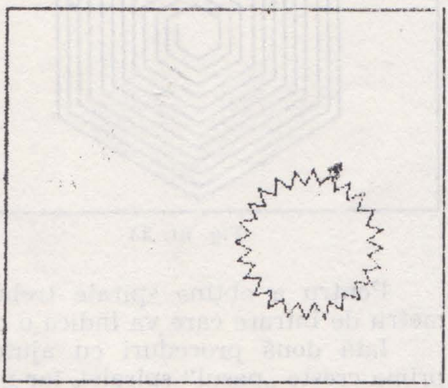


Fig. nr. 32

```
TO POLI :LAT :UNGHI
FD :LAT RT :UNGHI
POLI :LAT :UNGHI
END
```

Experimentați procedura POLI pentru mai multe valori ale variabilelor și încercați să vă imaginați dinainte cum vor arăta rezultatele pe ecran. În fig. 30 puteți observa rezultatul obținut prin POLI 70 160.

Modificați procedura POLI adăugînd înainte de apelul recursiv instrucțiunile :

FD :LAT * 2 RT :UNGHI

Experimentați din nou procedura pentru mai multe valori ale variabilelor și încercați să vă imaginați dinainte cum vor arăta rezultatele pe ecran. În fig. 31 puteți observa rezultatul obținut prin POLI 30 150.

Modificaţi procedura POLI înlocuind instrucţiunile adăugate cu următoarele :

FD : LAT RT : UNGHI * 2

Observaţi că de data aceasta nu mai creşte latura, ci unghiul.

Experimentaţi procedura POLI pentru mai multe valori ale variabilelor. În fig. 32 puteţi observa rezultatul obţinut prin POLI 8 125.

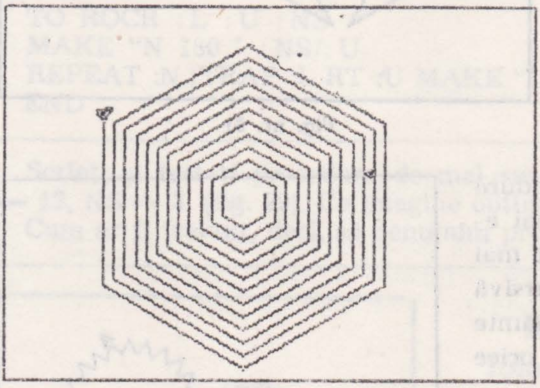


Fig. nr. 33

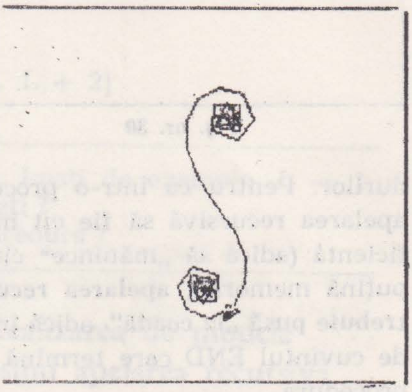


Fig. nr. 34

Pentru a obţine spirale trebuie adăugat procedurii încă un parametru de intrare care va indica o creştere.

Iată două proceduri cu ajutorul cărora se pot obţine spirale (în prima creşte „pasul" spiralei, iar în a doua, unghiul):

```
TO SPIRAL1 :PAS :UNGHI :CREŞTERE
FD : PAS RT : UNGHI
SPIRAL1 : PAS + : CREŞTERE : UNGHI : CREŞTERE
END
```

Experimentaţi procedura pentru mai multe valori.

În fig. 33 puteţi observa rezultatul obţinut prin :

SPIRAL1 10 60 1

```
TO SPIRAL2 : PAS : UNGHI : CREŞTERE
FD : PAS RT : UNGHI
SPIRAL2 : PAS : UNGHI + : CREŞTERE : CREŞTERE
END
```

Experimentaţi procedura pentru mai multe valori. În fig. 34 puteţi observa rezultatul obţinut prin :

SPIRAL2 12 100 8.

JOCURI

Puzzle

Un pătrat cu latura de 50 este divizat în 7 părţi care reprezintă figuri geometrice diverse (vezi fig. 35). Figurile astfel obţinute (numerotate de la 1 la 7) se pot combina în nenumărate moduri. Jocul constă în combinarea celor 7 figuri geometrice pentru a forma diverse desene. Să vedem cum utilizăm figurile pentru a desena un cîine. Sigur, va trebui să realizăm cîte o procedură pentru fiecare figură şi, deasemenea, să integrăm aceste proceduri într-una principală numită CÎINE care le va apela şi le va aranja (combina) în aşa fel încît să rezulte desenul unui cîine. Broasca trebuie poziţionată în poziţie centrală înaintea fiecărei trasări a unei figuri.

Iată procedurile pentru cele 7 figuri de bază precum şi procedura (programul) care desenează un cîine :

Pentru figura 1 (un pătrat mic)

```
TO PAT
REPEAT 4 [FD 25 RT 90]
END
```

Pentru figura 2 (un paralelogram)

```
TO PRIN
REPEAT 2 [FD 25 RT 45 FD 35 RT 135]
END
```

Pentru figurile 3 şi 4 (triunghiuri mici cu un unghi drept, identice dar puse în poziţii diferite)

```
TO TRIMIC
FD 25 RT 135 FD 35
RT 135 FD 25 RT 90
END
```

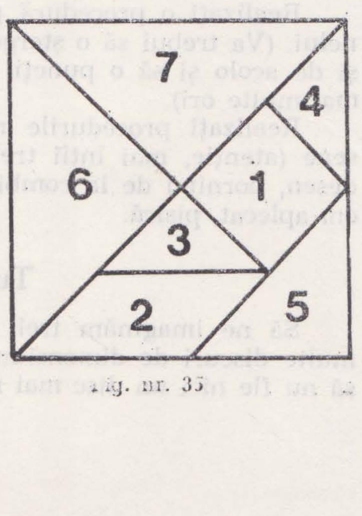


Fig. nr. 35

Pentru figura 5 (un triunghi mediu, cu un unghi drept)

```
TO TRIMED
FD 35 RT 135 FD 50
RT 135 FD 35 RT 90
END
```

Pentru figurile 6 și 7 (triunghiuri mari identice, dar în poziții diferite)

```
TO TRIMAR
FD 50 RT 135 FD 71
RT 135 FD 50 RT 90
END
```

Iată și programul de desenare al cîinelui :

```
TO CÎINE
TRIMAR DEPL1 PRIN DEPL2
TRIMED DEPL3 TRIMIC DEPL4
TRIMAR DEPL5 TRIMIC DEPL6
PAT DEPL7
END
```

Procedurile de deplasare a figurilor :

```
TO DEPL1
PU FD 15 LT 45 PD
END

TO DEPL2
PU RT 45 FD 35
LT 45 BK 35 PD
END

TO DEPL3
PU LT 45 BK 25 PD
END

TO DEPL4
PU RT 90 BK 25 PD
END

TO DEPL5
PU FD 50 RT 45 PD
END

TO DEPL6
PU FD 25 RT 135
FD 5 LT 90 PD
END

TO DEPL7
PU LT 90 FD 5
RT 45 BK 25 FD 45
BK 50 LT 90 BK 50 PD
END
```

Realizați o procedură prin intermediul căreia să mișcați coada cîinelui. (Va trebui să o ștergeți și să o desenați alături, apoi să o ștergeți și de acolo și să o puneți la loc, iar acest procedeu trebuie repetat de mai multe ori).

Realizați procedurile necesare pentru realizarea următoarelor desene (atenție, mai întîi trebuie rezolvată problema compunerii fiecărui desen, pornind de la combinarea celor 7 figuri) : om fugind, om așezat, om aplecat, pisică.

Turnurile din Hanoi

Să ne imaginăm trei vergele A, B și C, iar pe vergeaua A mai multe discuri de dimensiuni diferite unele peste altele, în așa fel încît, să nu fie nici un disc mai mare peste unul mai mic. Scopul jocului este

de a muta întreaga stivă de discuri de pe vergeaua A (să o numim sursă) pe vergeaua C (să o numim destinaţie), putînd fi folosită în acest scop şi vergeaua B (să o numim intermediar) (vezi fig. 36). Regula jocului este de a muta numai disc mic peste mare, nu şi invers, iar mutarea întregii stive de discuri să se facă dintr-un număr cît mai mic de mutări. Transpunerea acestui joc pe calculator este relativ simplă :

— pentru fiecare mutare se testează legalitatea ei ;

— dacă mutarea nu este legală se adaugă la numărul de mutări o unitate şi se dă posibilitatea jucătorului de a efectua o nouă mutare ;

— dacă mutarea este legală, atunci se efectuează, iar apoi se testează dacă poziţia la care s-a ajuns este cea finală, adică dacă toate discurile sînt pe vergeaua de destinaţie ;

— dacă poziţia la care s-a ajuns nu este cea finală, atunci numărul de mutări creşte cu o unitate şi se oferă jucătorului posibilitatea de a efectua o nouă mutare ;

— dacă poziţia la care s-a ajuns este chiar cea finală, atunci jocul s-a terminat şi se afişează mesajul de terminare şi numărul de mutări în care s-a ajuns la poziţia finală.

Mai interesant decît de construit acest joc este de a realiza un program care să rezolve problema acestui joc, adică de a realiza un program pentru jucător. Să vedem cum se rezolvă problema pentru cazul cel mai simplu, în care avem 3 discuri :

1. se mută discul 1 pe vergeaua C ;
2. se mută discul 2 pe vergeaua B ;
3. **se mută discul 1 pe vergeaua B (deasupra discului 2) ;**
4. **se mută discul 3 pe vergeaua C ;**
5. se mută discul 1 pe vergeaua A ;
6. se mută discul 2 pe vergeaua C (deasupra discului 3) ;
7. **se mută discul 1 pe vergeaua C (deasupra discului 2).**

Observaţi că problema se poate rezolva din minimul 8 (2 ↑ N—1) mutări. (↑ este semnul pentru ridicare la putere).

Pentru mai multe discuri problema se complică, însă se va simplifica întrezărind posibilitatea de rezolvare identificînd pentru orice număr de discuri următoarele mutări şi poziţii cheie (vezi fig. 37) :

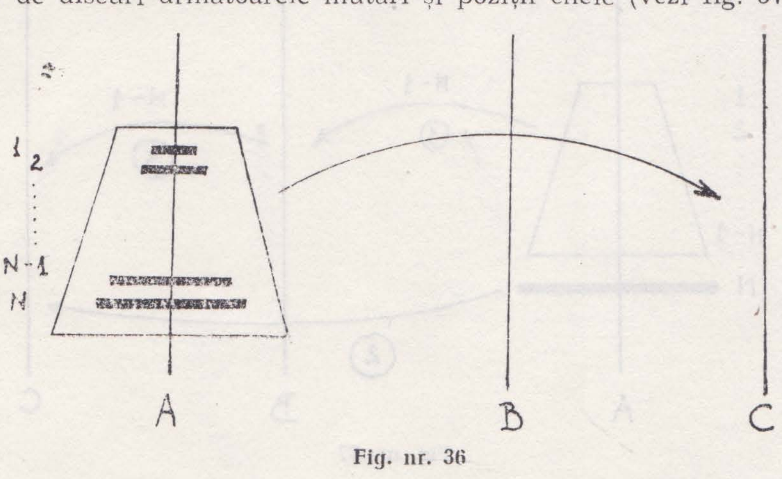


Fig. nr. 36

1. se mută stiva de N—1 discuri de deasupra discului N (baza) pe vergeaua B (intermediară) (vezi mutarea 1 din fig. 37). Această poziție se identifică cu cea la care s-a ajuns la pasul 3 în cazul rezolvării problemei pentru 3 discuri ;

2. se mută discul N (baza) pe vergeaua C (vezi mutarea 2 din fig. 37). Această poziție se identifică cu cea la care s-a ajuns la pasul 4 în cazul rezolvării problemei pentru 3 discuri ;

3. se mută stiva de N—1 discuri de pe vergeaua B (intermediară) pe vergeaua C (destinația) (vezi mutarea 3 din fig. 37). Această poziție se identifică cu cea la care s-a ajuns la pasul 7 în cazul rezolvării problemei pentru 3 discuri și este, de fapt, poziția finală. Cum veți muta, însă, stiva de N-1 discuri de pe vergeaua A pe vergeaua B (mutarea 1) și apoi de pe vergeaua B pe vergeaua C ? Simplu ! Dacă am putut muta o stivă de N discuri de pe vergeaua A pe vergeaua C, vom putea muta și una de N-1 discuri de pe vergeaua A pe vergeaua B folosind un apel recursiv de procedură, în care numărul de discuri precum și vergelele (sursă, intermediar și destinație) vor fi altele.

Realizați un program care să rezolve problema turnurilor din Hanoi pentru orice număr de discuri astfel :

— realizați o procedură principală (Hanoi) de 4 parametri : primul, numărul de discuri (N) iar următoarele 3, vergelele sursa, intermediarul și destinația (în această ordine) ;

— dacă numărul de discuri este 0 atunci procedura se oprește ;

— se apelează procedura principală pentru N-1 discuri, dar de data aceasta vergeaua sursă este A, destinația este B, intermediar C (mutarea 1 din fig. 37) ;

— se apelează o procedură (MUTĂDISC) care mută discul N de pe vergeaua A pe vergeaua C (mutarea 2 din fig. 37) ;

— se apelează procedura principală pentru N-1 discuri, dar de data aceasta vergeaua sursă este B, destinația este C, iar intermediar A (mutarea 3 din fig. 37) ;

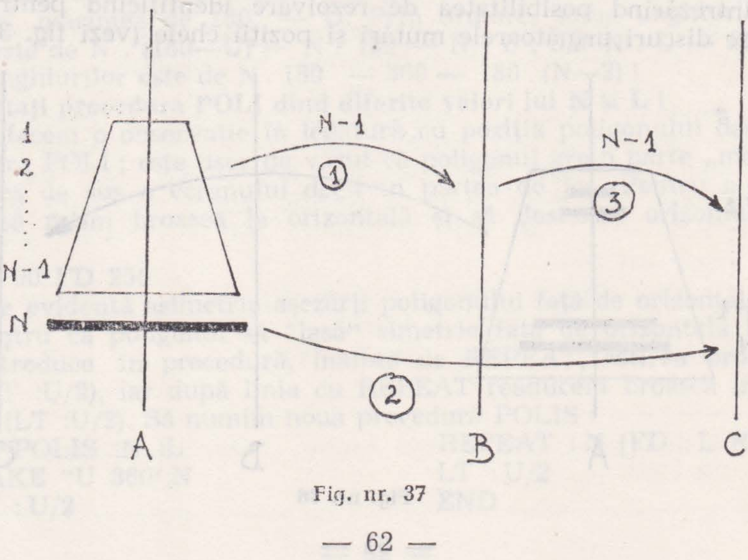


Fig. nr. 37

— se realizează procedura MUTĂDISC de 3 parametri (numărul discului de mutat, vergeaua pe care se găsește și vergeaua pe care trebuie să ajungă) care să realizeze mutarea discului indicat de pe o vergea pe alta.

Pentru cei care întîmpină greutăți în scrierea procedurilor iată un exemplu:

```
TO HANOI :N :A :B :C
IF :N=0 [STOP]
HANOI :N—1 :A :C :B
MUTĂDISC :N :A :C
HANOI :N—1 :B :A :C
END
```

```
TO MUTĂDISC :N :A :B
PR SE [SE MUTĂ DISCUL] :N
PR SE SE SE [DE PE VERGEAUA]
:A [PE VERGEAUA] :B
END
```

De ce avem nevoie de trei SENTENCE? Deoarece trebuie concatenate 4 obiecte, iar SE necesită doar 2 parametri de intrare. Deci primul SE (cel mai din dreapta) va concatena primii doi parametri, al doilea va concatena rezultatul cu următorul obiect (al treilea) și, în sfîrșit, ultimul SE (cel mai din stînga) va concatena rezultatul obținut cu ultimul parametru, obținîndu-se o propoziție de tipul: „DE PE VERGEAUA A PE VERGEAUA C".

Pentru a vă juca cu Hanoi apelați, de exemplu, HANOI 10 "A "B "C și veți vedea ce repede rezolvă LOGO, cu un program făcut de noi, problema turnurilor din Hanoi cu 10 discuri în care vergeaua sursă se numește A, intermediara B, iar sursa C.

Fractali

Ne propunem să realizăm o procedură cu ajutorul căreia să desenăm un copac. Spre deosebire de proiectul de desenare a unei case, problema, în acest caz, este de a identifica o metodă recursivă, adică de a concepe copacul prin părți mai mici dar similare cu cea inițială. La prima vedere problema s-ar putea rezolva dacă am considera copacul format dintr-un trunchi din care se desprind două ramuri mai mici, iar apoi din fiecare ramură se desprind alte două subramuri (crengi), și așa mai departe (vezi fig. 38).

O primă aproximare a soluției în care subramurile să fie jumătăți din ramurile din care provin ar putea arăta astfel :

```
TO COPAC1 :MĂRIME
FD :MĂRIME LT 20
COPAC1 :MĂRIME/2 RT 40
COPAC1 :MĂRIME/2
END
```

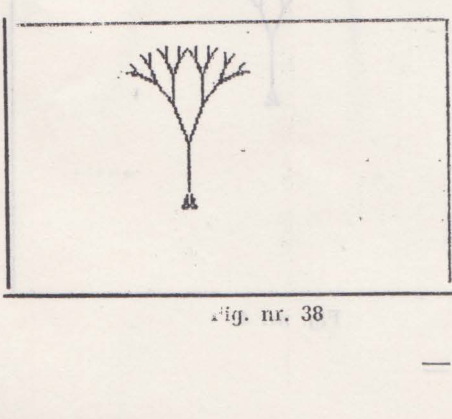


Fig. nr. 38

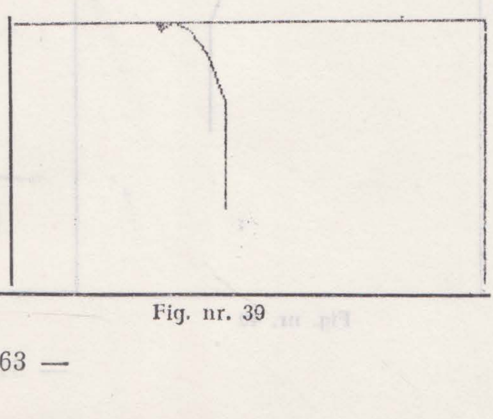


Fig. nr. 39

Un rezultat al acestei proceduri se poate observa în fig. 39. În această formă desenul nu prea ne mulțumește, iar faptul că în procedură nu este implicată o regulă de oprire are ca efect autoapelarea procedurii la nesfîrșit și, deci, desenarea de ramuri din ce în ce mai mici. Care ar putea fi, în acest caz, o condiție de limitare ?

Ar putea exista două posibilități de limitare a numărului de ramuri ale copacului. Prima posibilitate se poate materializa prin indicarea explicită a numărului de invocații recursive. În acest caz vom adăuga un alt parametru (:NIR) care va indica numărul de invocații recursive care vor fi permise. Acum noua procedură pentru COPAC precum și rezultatul său pe ecran vor arăta astfel (vezi fig. 40) :

```
TO COPAC2 : NIR : MAR
IF : NIR = 0 [STOP]
FD : MAR LT 20
Un.V6TT3ch-,,B 5.toDA
COPAC2 ( : NIR — 1) : MAR/2
RT 40
COPAC2 ( : NIR — 1) : MAR/2
END
```

O a doua posibilitate (impusă mai ales de faptul că încă nu sîntem mulțumiți de desen) este de a se invoca apelurile recursive pînă cînd crengile ating un minim rezonabil :

```
TO COPAC3 : MAR
IF : MAR < 4 [STOP]
FD : MAR LT 20
COPAC3 : MAR/2
RT 40
COPAC3 :MAR/2
LT 20
BK : MAR
END
```

În fig. 41 se redă rezultatul obținut cu această procedură (cu valoarea 30 a parametrului de intrare) iar în fig. 42 cel mai mulțumitor rezultat de pînă acum (COPAC4) obținut cu valoarea 30 a parametrului de intrare și prin înlocuirea valorii expresiei :MAR/2 din apelul recursiv cu expresia :MAR * 2/3. Prin această modificare ramurile rezultante nu vor mai fi jumătăți ale ramurii din care provin ci 2/3 din ea, adică nu își vor mai atinge minimul așa de repede, iar coroana copacului va fi mai bogată.



Fig. nr. 40

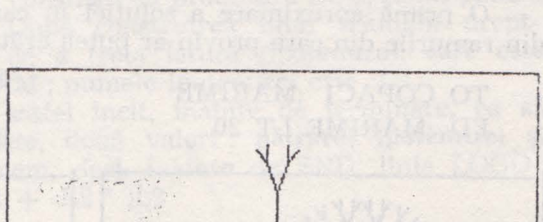


Fig. nr. 41



Fig. nr. 42



Fig. nr. 43

Se observă că ultimele două proceduri (COPAC3 și COPAC 4) mai prezintă încă o diferență față de cele precedente și anume adăugarea înainte de sfîrșitul procedurii a încă două instrucțiuni :LT 20 și BK :MAR. Această adăugare a fost necesară deoarece în vechea versiune broasca nu avea la sfîrșitul procedurii aceeași poziție și direcție pe care le avea inițial. Pentru a îndrepta acest lucru a fost necesară reîntoarcerea broaștei la baza ramurii pentru fiecare subramură.

Intr-adevăr ultimele versiuni produc copaci rezonabili. Deoarece pentru oprire s-a ales un minim de ramuri, forma copacului depinde de mărimea trunchiului, ceea ce este destul de diferit față de practica uzuală prin care se obișnuiește desenarea aceleiași forme corespunzătoare unei anumite mărimi.

Explorarea recursivă a unor forme cum este și cea a copacului și legarea acesteia de utilizări practice este de dată relativ recentă. Matematicianul Benoit Mandelbrot a dat numele de **fractali** figurilor recursive și a fost primul care a văzut în ele utilizări practice importante (în lucrarea **„Geometria fractală în natură"** în anul 1982). Firma Lucasfilm, de exemplu, a utilizat pe larg grafica pe calculator bazată pe fractali, ca o alternativă la modelele foarte costisitoare în realizarea efectelor speciale în filme science fiction. S-a dovedit că programe (cum este chiar COPAC) au reprezentat metode eficiente de realizare (desenare) a unor scenarii.

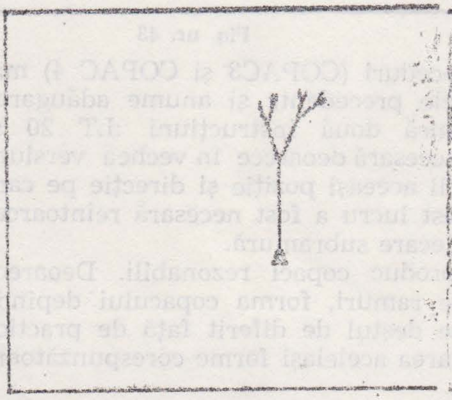
Sigur, problema obținerii unor desene realiste este mai complexă. Trebuie să mărturisim că ultimii copaci obținuți sînt prea uniformi și simetrici, ceea ce nu corespunde, de obicei, realității din natură, În cazul nostru soluția ar fi să se permită ceva întîmplător în alegerea mărimilor ramurilor și a unghiurilor.

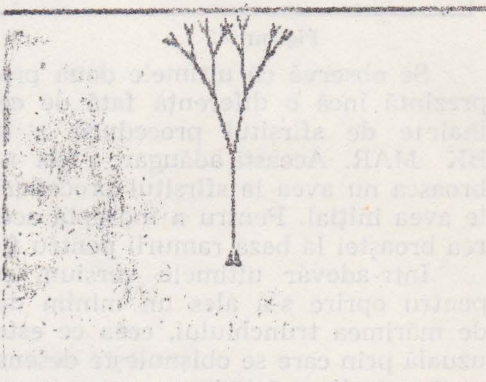
Prezentăm în continuare o soluție pentru rezolvarea problemei. Definim o operație RANG al cărei rezultat este un număr egal cu parametrul de intrare (dacă acesta a fost un număr) sau un număr cuprins într-un interval (dacă parametrul de intrare a fost o listă formată din cele două numere care determină intervalul) :

```
TO RANG :RANG
IF WORDP :RANG [OUTPUT :RANG]
OUTPUT RANG1 FIRST :RANG LAST :RANG
END
```

```
TO RANG1 :DE.LA :LA
OUTPUT SUM :DE.LA RANDOM 1 + :LA — :DE.LA
END
```

În acest caz programul pentru desenarea copacului va conţine două proceduri (COPAC 5 şi COPAC 6) care se vor apela una pe alta. Procedura principală (COPAC5) are patru parametri care reprezintă : mărimea trunchiului principal al copacului (primul parametru), procentul (raţia) cu care fiecare ramură este mai mică faţă de cea desenată înaintea ei (de exemplu, al doilea parametru 50 va face fiecare ramură jumătate faţă de cea din care provine), unghiul cu care se roteşte broasca înainte de desenarea subramurii din stînga şi unghiul cu care se roteşte broasca la dreapta pentru desenarea celei de a doua subramuri parametrii 3 şi, respectiv, 4).


Fig. nr. 44


Fig. nr. 45

```
TO COPAC5 :M :R :L :U
COPAC6 (RANG :M) (RANG :L) (RANG :U)
END
TO COPAC6 :MAR :LTURN :RTURN
IF :MAR < 4 [STOP]
FD :MAR LT :LTURN
COPAC5 (:MAR * (RANG :R)/100) :R :L :U
RT :LTURN + :RTURN
COPAC5 (:MAR * (RANG :R)/100) :R :L :U
LT :RTURN
BK :MAR
END
```

Iată şi desenele obţinute : pentru COPAC5 50 60 [10 30] [5 25] vezi fig. 43 ; pentru COPAC5 [30 50] [30 60] 15 25 (vezi fig. 44) şi pentru COPAC5 [30 50] [30 60] [10 30] [5 25] (vezi fig.45). Se observă că pentru ultimele două desene, mărimea iniţială a trunchiului copacului poate fi orice număr (aleator) cuprins între 30 şi 50.

Realizaţi un fractal pentru desenarea unui fulg de nea.

MEMORATOR LOGO:

LISTA COMPLETĂ A COMENZILOR ŞI OPERAŢIILOR

LEGENDA:

— În cadrul unei grupe comenzile şi operaţiile sînt evidenţiate în ordine alfabetică ;

— În paranteză : comenzi în formă prescurtată ;

— Pe aceeaşi linie : comenzi echivalente ;

— Comenzile în limba română (coloana 2) sînt accesibile dacă s-a încărcat în prealabil fişierul ROMANA ;

Notate cu semnul * : comenzi sau operaţii accesibile numai pentru configuraţii cu disc flexibil.

1. Broasca ţestoasă

Comenzi originale	Comenzi în limba română	Necesită subiect (parametru de intrare)
1	2	3
BACK (BK)	ÎNAPOI (ÎP)	da
BACKGROUND (BG)	FOND	—
CLEAN	—	—
CLEARSCREEN (CS)	ŞTERGE	—
DOT	PUNCT	da
FENCE	GARD	—
FORWARD (FD)	ÎNAINTE (ÎN)	da
HEADING	DIRECŢIE	—
HIDETURTLE (HT)	FĂRĂBROASCĂ (FB)	—
HOME	ACASA	—
LEFT (LT)	STÎNGA (SA)	da
PENCOLOUR (PC)	CULOARE	—
PENDOWN (PD)	CREION (CR)	—
PENERASE (PE)	GUMĂ (GU)	—
PENREVERSE (PX)	CI	—
PENUP (PU)	FĂRĂCREION (FC)	—
POSITION (POS)	POZIŢIE	—
RIGHT (RT)	DREAPTA (DR)	da
SCRUNCH	RELXY	—
SETBG	FIXFOND	da

SETBORDER (SETBR)	FIXCHENAR	da
SETHEADING (SETH)	FIXDIR	da
SETPC	FIXCULOARE	da
SETPOS	FIXPOZ	da
SETSCRUNCH (SETSCR)	FIXSCARA	da
SETX	FIXX	da
SETY	FIXY	da
SHOWNP	—	—
SHOWTURTLE (ST)	BROASCA	—
TOWARDS	CAP	da
WINDOW	FEREASTRĂ	—
WRAP	IESEREVINE	—
XCOR	—	—
YCOR	—	—

2. Cuvinte şi liste

1	2	3
ASCII	—	da
BUTFIRST (BF)	FĂRĂPRIMUL (FP)	da
BUTLAST (BL)	FĂRĂULTIMUL (FU)	da
CHAR	CARACTER	da
COUNT	NREAL	da
EMPTYP	GOL	da
EQUALP	EGAL	da
FIRST	PRIMUL	da
FPUT	PUNEPRIMUL	da
ITEM	—	da
LAST	ULTIMUL	da
LIST	LISTĂ	da
LISTP	—	da
LPUT	PUNEULTIMUL	da
MEMBERP	—	da
NUMBERP	—	da
SENTENCE (SE)	FRAZA	da
WORD	—	da
WORDP	—	da

3. Variabile

1	2	3
MAKE	PUNE	—
NAMEP	—	—
THING	—	—

4. Operații aritmetice

1	2	3
ARCCOS	—	da
ARCCOT	—	da
ARCSIN	—	da
ARCTAN	—	da
COSINE (COS)	—	da
COTANGENT (COT)	—	da
DIV	—	da
INT	—	da
PRODUCT	PROD	da
RANDOM	—	da
REMAINDER	REST	da
ROUND	—	da
SINE (SIN)	—	da
SQRT	—	da
SUM	—	da
TANGENT (TAN)	—	da
+ — * / =	—	da

5. Definiri și editări de proceduri

1	2	3
EDIT (ED)	—	da/nu
EDNS	—	da
END	—	—
TO	—	da

6. Condiții și control

1	2	3
BYE	—	—
IF	DACĂ	da
OUTPUT (OP)	—	da
REPEAT	REPETĂ	da
RUN	—	da
STOP	—	—
TOPLEVEL	—	—

7. Operații logice

1	2	3
AND	ȘI	da
FALSE	FALS	da

NOT	NU	da
OR	SAU	da
TRUE	ADEVĂRAT	da

8. Extragere rezultate

1	2	3
KEYP	—	—
PRINT (PR)	SCRIE	da
READCHAR (RC)	CITCAR	—
READLIST (RL)	CITLIST	—
SHOW	—	—
STARTROBOT	—	—
SOUND	SUNET	da
STOPROBOT	—	—
TYPE	—	da
WAIT	AȘTEAPTĂ	da

9. Ecran

1	2	3
BRIGHT	—	da
CLEARTEXT	ȘTERGETEXT	—
CURSOR	—	—
FLASH	—	—
INVERSE	—	—
NORMAL	—	—
OVER	—	da
SETCURSOR (SETCUR)	FIXCURSOR	da
SETTC	—	da
TEXTCOLOUR (TC)	—	—
TEXTSCREEN (TS)	—	—

10. Spațiul de lucru

1	2	3
ERALL	—	—
ERASE (ER)	UITA	da
ERN	—	da
ERNS	—	—
ERPS	—	—
PO	—	da
POALL	—	—
PONS	—	—
POPS	—	—
POTS	—	—

11. Salvări şi încărcări

1	2	3
CATALOG*	—	—
COPYSCREEN*	—	—
ERASEFILE*	—	da
LOAD	—	da
LOADD	—	da
LOADSCR	—	da
PRINTON	—	—
PRINTOFF	—	—
SAVE	—	da
SAVEALL	—	da
SAVED	—	da
SAVESCR	—	da
SETDRIVE*	—	da

12. Definiri şi redefiniri de funcţii

1	2	3
COPYDEFF	—	da
DEFINE	—	da
DEFINEDP	—	da
PRIMITIVEP	—	da
TEXT	—	—

13. Primitive avansate

1	2	3
NODES	—	—
RECYCLE	—	—
. BLOAD	—	da
. BSAVE	—	da
. CALL	—	da
. CONTENTS	—	—
. DEPOSIT	—	da
. EXAMINE	—	da
. PRIMITIVES	—	—
. RESERVE	—	da
. RESERVED	—	—
SERIALIN	—	—
. SERIALOUT	—	da
. SETSERIAL	—	da

CUPRINS

Tiparul executat sub c-da nr. 21/1990
la I. P. „Filaret" III

I.S.B.N. 973-95036-1-6

LEI 43